Oracle® Grid Engine

User Guide

Release 6.2 Update 7

E21976-01

August 2011

Oracle Grid Engine User Guide, Release 6.2 Update 7

E21976-01

Primary Author: Uma Shankar

Contributing Author:

Contributor:

Contents

Preface vii

Audience vii
Documentation Accessibility vii
Related Documents vii
Conventions vii

1 Getting Started

1.1 How the System Operates 1-1
1.2 How Resources Are Matched to Requests 1-4
1.3 A Banking Analogy 1-4
1.3.1 Jobs and Queues 1-5
1.4 Usage Policies 1-5
1.4.1 Using Tickets to Administer Policies 1-6
1.4.2 Using the Urgency Policy to Assign Job Priority 1-6
1.5 Choosing a User Interface 1-7
1.5.1 QMON - The Graphical User Interface 1-7
1.5.2 The Command Line Interface 1-8
1.5.3 The Distributed Resource Management Application API (DRMAA) 1-8
1.6 Users and User Categories 1-8

2 Using Grid Engine

2.1 Interacting With Grid Engine as a User 2-1
2.1.1 Launching QMON From the Command Line 2-1
2.1.2 Customizing QMON 2-2
2.1.3 Using the Command-Line Interface 2-2
2.2 Displaying User Properties 2-2
2.2.1 User Access Permissions 2-3
2.2.2 Displaying Managers, Operators, Owners, and User Access Permissions 2-4
2.3 Displaying Host Properties 2-5
2.4 Displaying Queue Properties 2-6
2.4.1 Interpreting Queue Property Information 2-6
2.5 Submitting Jobs 2-7
2.5.1 How Jobs Are Scheduled 2-7
2.5.2 Usage Policies 2-8
2.5.3 Job Priorities 2-9

2.5.4 Ticket Policies 2-9
2.5.5 Queue Selection 2-9
2.5.6 Defining Resource Requirements 2-10
2.5.7 Requestable Attributes 2-12
2.6 Submitting Batch Jobs 2-15
2.6.1 About Shell Scripts 2-15
2.6.2 Extensions to Regular Shell Scripts 2-16
2.6.2.1 How a Command Interpreter is Selected 2-16
2.6.2.2 Output Redirection 2-16
2.6.2.3 Active Comments 2-17
2.6.2.4 Environment Variables 2-18
2.7 Submitting Array Jobs 2-20
2.7.1 How to Configure Array Task Dependencies From the Command Line 2-20
2.7.2 How to Submit an Array Job From the Command Line 2-23
2.7.3 How to Submit an Array Job With QMON 2-23
2.8 Submitting Interactive Jobs 2-23
2.8.1 How to Submit Interactive Jobs From the Command Line 2-24
2.8.1.1 Using qrsh to Submit Interactive Jobs 2-24
2.8.1.2 Using qsh to Submit Interactive Jobs 2-25
2.8.1.3 Using qlogin to Submit Interactive Jobs 2-25
2.8.2 How to Submit Interactive Jobs With QMON 2-26
2.9 Transparent Remote Execution 2-27
2.9.1 Remote Execution With qrsh 2-27
2.9.1.1 Invoking Transparent Remote Execution With qrsh 2-28
2.9.2 Transparent Job Distribution With qtcsh 2-28
2.9.2.1 qtcsh Usage 2-29
2.9.3 Parallel Makefile Processing With qmake 2-30
2.9.3.1 qmake Usage 2-31
2.10 How to Submit a Simple Job From the Command Line 2-32
2.11 How to Submit a Simple Job With QMON 2-34
2.12 How to Submit an Extended Job From the Command Line 2-36
2.13 How to Submit an Extended Job With QMON 2-36
2.14 How to Submit an Advanced Job From the Command Line 2-39
2.14.1 Specifying the Use of a Script or a Binary 2-40
2.14.2 Default Request Files 2-40
2.15 How to Submit an Advanced Job With QMON 2-41
2.16 How to Configure Job Dependencies From the Command Line 2-43
2.17 Monitoring Hosts from the Command Line 2-43
2.17.1 Using qconf 2-43
2.17.2 Using qhost 2-43
2.18 How to Monitor Hosts With QMON 2-44
2.18.1 Hosts Status 2-44
2.19 Monitoring and Controlling Jobs 2-45
2.19.1 How to Monitor Jobs From the Command Line 2-45
2.19.2 How to Monitor Jobs With QMON 2-47
2.19.2.1 How to Get Additional Information About Jobs With the QMON Object Browser . 2-48
2.19.3 How to Control Jobs From the Command Line 2-48

2.19.4 How to Control Jobs With QMON 2-49
2.19.5 How to Monitor Jobs by Email 2-52
2.19.6 How to Monitor Jobs by Email With QMON 2-53
2.20 Monitoring and Controlling Queues 2-53
2.20.1 How to Control Queues From the Command Line 2-53
2.20.2 How to Monitor and Control Cluster Queues With QMON 2-54
2.20.2.1 Cluster Queue Status 2-54
2.20.3 How to Monitor Queues With QMON 2-55
2.21 Using Job Checkpointing 2-56
2.21.1 Migrating Checkpointing Jobs 2-56
2.21.2 File System Requirements for Checkpointing 2-56
2.21.3 Writing a Checkpointing Job Script 2-56
2.21.4 How to Submit a Checkpointing Job From the Command Line 2-57
2.21.5 How to Submit a Checkpointing Job With QMON 2-58
2.22 Managing Core Binding 2-58
2.22.1 Submit Simple Jobs with Core Binding 2-58
2.22.2 Submit Array Jobs with Core Binding 2-58
2.22.3 Submit Parallel Jobs with Core Binding 2-59
2.22.3.1 Submit Tightly Integrated Parallel Jobs with Core Binding 2-60
2.23 Automating Grid Engine Functions Through DRMAA 2-60
2.23.1 Developing With the C Language Binding 2-60
2.23.1.1 Important Files for the C Language Binding 2-61
2.23.1.2 Including the DRMAA Header File 2-61
2.23.1.3 Compiling Your C Application 2-61
2.23.1.4 Running Your C Application 2-61
2.23.1.5 C Application Examples 2-62
2.23.2 Developing With the Java Language Binding 2-65
2.23.2.1 Important Files for the Java Language Binding 2-65
2.23.2.2 Importing the DRMAA Java Classes and Packages 2-65
2.23.2.3 Compiling Your Java Application 2-65
2.23.2.4 How to Use DRMAA With NetBeans 5.*x* 2-65
2.23.2.5 Running Your Java Application 2-66
2.23.2.6 Java Application Examples 2-66
2.24 Using the Accounting and Reporting Console 2-68
2.25 Installing the Accounting and Reporting Console (ARCo) 2-69
2.25.1 Configuring the Database Server 2-69
2.25.2 How to Configure the ARCo Database on MySQL 2-69
2.25.3 How to Configure the ARCo Database on PostgresSQL 2-70
2.25.4 How to Configure the ARCo Database with Multiple Schemas on PostgresSQL . 2-71
2.25.5 How to Configure the MySQL Database Server 2-73
2.25.5.1 MySQL Installation Tips 2-73
2.25.5.2 Case Sensitivity in MySQL Database 2-73
2.25.6 How to Configure the PostgresSQL Server 2-75
2.25.7 Using the Oracle Database 2-77
2.25.8 How to Add Authorized ARCo Users 2-78
2.25.9 How to Install dbwriter 2-78
2.25.10 Example dbwriter Installation 2-82

2.25.11 How to Install Reporting 2-86
2.25.12 Example Reporting Installation 2-88
2.25.13 How to Install Sun Java Web Console 2-92
2.26 Planning the ARCo Installation 2-93
2.26.1 Supported Operating Platforms 2-94
2.26.2 Required Software 2-94
2.26.3 Disk Space Recommendations 2-95
2.26.4 Multi-Cluster Support Overview 2-95
2.26.5 Database Configuration Illustrations 2-95
2.26.6 Schema Overview 2-97
2.27 How to Start ARCo 2-98
2.27.1 How to Start the Accounting and Reporting Console 2-98
2.27.2 Creating and Modifying Simple Queries 2-99
2.27.3 Creating and Modifying Advanced Queries 2-102
2.27.4 Configuring the Query Results View 2-104
2.27.5 Examples for Defining Graphical Views 2-107
2.28 ARCo Configuration Files and Scripts 2-109
2.28.1 About dbwriter 2-110
2.28.1.1 inst_dbwriter Command Options 2-110
2.28.1.2 dbwriter Configuration Parameters 2-110
2.28.1.3 sgedbwriter Command Options 2-111
2.28.2 About Reporting 2-112
2.28.3 Other ARCo Utilities 2-114
2.29 Creating Cross-Cluster Queries 2-115
2.30 Examples 2-116
2.30.1 Example - arco_write_london.sge_user 2-116
2.30.2 Example - arco_write_denver.sge_user 2-116
2.31 Derived Values and Deletion Rules 2-117
2.31.1 Derived Values 2-117
2.31.1.1 Derived Values Format 2-117
2.31.1.2 Derived Values Examples 2-118
2.31.2 Deleting Outdated Records 2-119
2.31.2.1 Deletion Rules Format 2-119
2.31.2.2 Deletion Rules Examples 2-120
2.32 ARCo Frequently-Asked Questions 2-121
2.33 ARCo Troubleshooting 2-122

3 Upgrading ARCO

3.1 How to Migrate a PostgreSQL Database to a Different Schema 3-1
3.2 How to Upgrade the ARCo Software 3-4

A Command Line Interface Ancillary Programs

A.1 List of Ancillary Programs A-1
A.2 User Access to the Ancillary Program A-2

Preface

The Oracle® Grid Engine User's Guide provides a description about Oracle Grid Engine architecture, system operation, and how to use the software to apply resource management strategies to distribute jobs across a grid

Audience

This document is intended for system administrators.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Grid Engine Release 6.2 documentation:

- *Oracle Grid Engine Release Notes*
- *Oracle Grid Engine Installation Guide*
- *Oracle Grid Engine Administration Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Getting Started

A grid is a collection of computing resources that performs tasks. In its simplest form, a grid appears to users as a large system that provides a single point of access to powerful distributed resources. In its more complex form, a grid can provide many access points to users.

The Grid Engine software enables you to apply resource management strategies to distribute jobs across a grid. Users can submit millions of jobs at a time without being concerned about where the jobs run. The system supports clusters with up to 63,000 cores.

Sites configure the system to maximize usage and throughput, while the system supports varying levels of timeliness and importance. Job priority and user share are instances of importance.

The Grid Engine software provides advanced resource management and policy administration for UNIX and Windows environments that are composed of multiple shared resources. For more on Grid Engine's features, see the product web site at:

`http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`

1.1 How the System Operates

The Grid Engine system does the following:

- **Accepts jobs** - Jobs are users' requests for computer resources. Each job includes a description of what to do and a set of property definitions that describe how the job should be run. Users can submit jobs via the command line interface or Grid Engine's graphical user interface, QMON. Users can also use the optional Distributed Resource Management Application API (DRMAA) to automate grid engine functions by writing scripts to submit and control jobs.
- **Holds jobs** - The Grid Engine master daemon holds jobs until the needed compute resources become available.
- **Sends** - When the compute resources become available, the master daemon sends the job to the appropriate execution host. The execution daemon on that host then executes the job.
- **Manages running jobs** - The master daemon manages running jobs. At a fixed interval, the master daemon receives reports from each execution daemon.
- **Logs the record of job execution when the jobs are finished** - The master daemon stores raw data. Users can also use the Accounting and Reporting Console (ARCo) to gather live reporting data from the Grid Engine system and to store the data for historical analysis in the reporting database, which is a standard SQL database.

Figure 1–1 Grid Engine System Operation

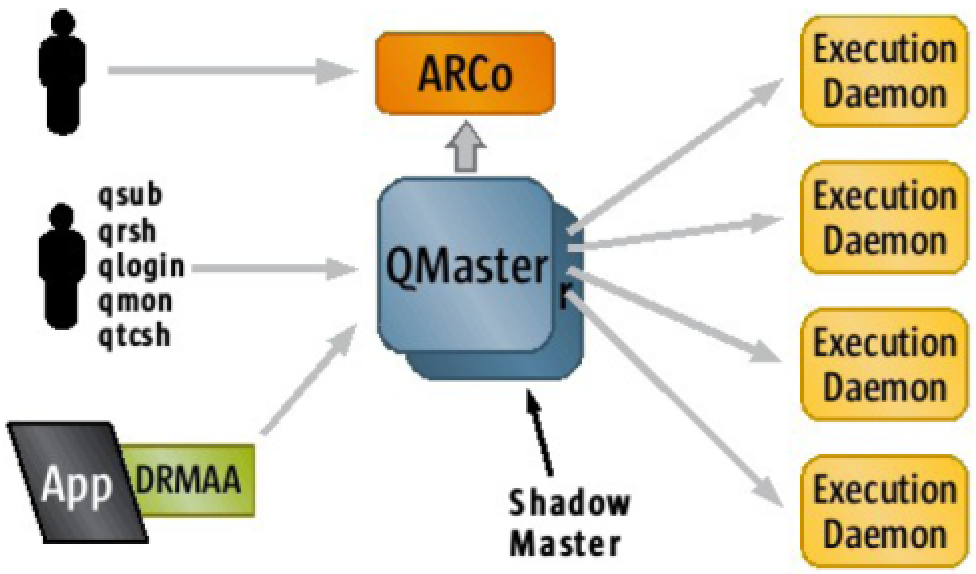


Table 1–1 Component Description

Component	Description	More Info
Cluster	A collection of machines, called hosts, on which Grid Engine system functions occur.	See CONFIGURING CLUSTERS.
Master Host	The master host is central to cluster activity. The master host runs the master daemon and usually also runs the scheduler. The master host requires no further configuration other than that performed by the installation procedure. By default, the master host is also an administration host and a submit host.	For information about how to initially set up the master host, see *Oracle Grid Engine Installation and Upgrade Guide* to install the master host. For information about how to configure dynamic changes to the master host, see *Oracle Grid Engine Administration Guide* for configuring hosts.
Master Daemon	The master daemon does the following: ▪ Accepts incoming jobs from users. ▪ Maintains tables about hosts, queues, jobs, system load, and user permissions. ▪ Performs scheduling functions and requests actions from execution daemons on the appropriate execution hosts. ▪ Decides which jobs are dispatched to which queues and how to reorder and reprioritize jobs to maintain share, priority, or deadline	See *Oracle Grid Engine Administration Guide* for configuring hosts.

Table 1–1 (Cont.) Component Description

Component	Description	More Info
Execution Host	Systems that have permission to run Grid Engine system jobs. These systems host queue instances, and run the execution daemon. Execution hosts are systems that have permission to execute jobs. Therefore, queue instances are attached to the execution hosts.	An execution host is initially set up by the installation procedure, as described in *Oracle Grid Engine Installation and Upgrade Guide* to install execution hosts. For installation planning guidance, see *Oracle Grid Engine Installation and Upgrade Guide* for host system requirements. See *Oracle Grid Engine Administration Guide* for more information on managing your cluster.
Execution Daemon	The execution daemon receives jobs from the master daemon and executes them locally on its host. An execution daemon is responsible for the queue instances on its host and for the running of jobs in these queue instances. Periodically, the execution daemon forwards information such as job status or load on its host to the master daemon.	See *Oracle Grid Engine Administration Guide* for configuring hosts.
Scheduler	The scheduler is responsible for prioritizing pending jobs and deciding which jobs to schedule to which resources.	For more information on the scheduler, see *Oracle Grid Engine Administration Guide* for managing the scheduler.
Administration Host	Administration hosts are hosts that have permission to carry out any kind of administrative activity for the Grid Engine system.	See *Oracle Grid Engine Administration Guide* for configuring hosts.
Submit Host	Submit hosts enable users to submit and control batch jobs only. In particular, a user who is logged in to a submit host can submit jobs with the `qsub` command, can monitor the job status with the `qstat` command, and can use the Grid Engine system OSF/1 Motif graphical user interface QMON, which is described in QMON, the Grid Engine System's Graphical User Interface.	See *Oracle Grid Engine Administration Guide* for configuring hosts.
Shadow Master Host	Shadow master hosts reduce unplanned cluster downtown. One or more shadow master hosts may be running on additional nodes in a cluster. In the case that the master daemon or the host on which it is running fails, one of the shadow masters will promote the host on which it is running to the new master daemon system by locally starting a new master daemon.	See *Oracle Grid Engine Installation and Upgrade Guide* for information about how to install the shadow master host.

Table 1–1 (Cont.) Component Description

Component	Description	More Info
DRMAA	The optional Distributed Resource Management Application API (DRMAA) automates Grid Engine functions by writing scripts that run Grid Engine commands and parse the results.	See Automating Grid Engine Functions Through DRMAA.
ARCo	The optional Accounting and Reporting Console (ARCo) enables you to gather live reporting data from the Grid Engine system and to store the data for historical analysis in the reporting database, which is a standard SQL database.	For more information, see Using the Accounting and Reporting Console.
SDM	The optional Service Domain Manager (SDM) module distributes resources between different services according to configurable Service Level Agreements (SLAs). The SLAs are based on Service Level Objectives (SLOs). SDM functionality enables you to manage resources for all kind of scalable services.	See SERVICE DOMAIN MANAGER for more information.

1.2 How Resources Are Matched to Requests

- A Banking Analogy
- Usage Policies

1.3 A Banking Analogy

As an analogy, imagine a large "money-center" bank in one of the world's capital cities. In the bank's lobby are dozens of customers waiting to be served. Each customer has different requirements. One customer wants to withdraw a small amount of money from his account. Arriving just after him is another customer, who has an appointment with one of the bank's investment specialists. She wants advice before she undertakes a complicated venture. Another customer in front of the first two customers wants to apply for a large loan, as do the eight customers in front of her.

Different customers with different needs require different types of service and different levels of service from the bank. Perhaps the bank on this particular day has many employees who can handle the one customer's simple withdrawal of money from his account. But at the same time the bank has only one or two loan officers available to help the many loan applicants. On another day, the situation might be reversed.

The effect is that customers must wait for service unnecessarily. Many of the customers could receive faster service if only their needs were immediately recognized and then matched to available resources.

If the Grid Engine system were the bank manager, the service would be organized differently:

- On entering the bank lobby, customers would be asked to declare their name, their affiliations, and their service needs.
- Each customer's time of arrival would be recorded.

- Based on the information that the customers provided in the lobby, the bank might serve the following customers in the following order:
  1. Customers whose needs match suitable and immediately available resources
  2. Customers whose requirements have the highest priority
  3. Customers who were waiting in the lobby for the longest time
- In a "Grid Engine system bank", one bank employee might be able to help several customers at the same time. The Grid Engine software would try to assign new customers to the least-loaded and most-suitable bank employee.
- As bank manager, the Grid Engine system would allow the bank to define service policies. Typical service policies might be the following:
  – To provide preferential service to commercial customers because those customers generate more profit

  – To make sure a certain customer group is served well, because those customers have received bad service in the past

  – To ensure that customers with an appointment get a timely response

  – To provide preferential treatment to certain customers because those customers were identified by a bank executive as high priority customers
- These policies would be implemented, monitored, and adjusted automatically by a Grid Engine system manager. Customers that have preferential access would be served sooner. Such customers would receive more attention from employees. The Grid Engine manager would recognize if the customers do not make progress. The manager would immediately respond by adjusting service levels to comply with the bank's service policies.

1.3.1 Jobs and Queues

In a Grid Engine system, jobs correspond to bank customers. Jobs wait in a computer holding area instead of a lobby. Queues, which provide services for jobs, correspond to bank employees. As in the case of bank customers, the requirements of each job, such as available memory, execution speed, available software licenses, and similar needs, can be very different. Only certain queues might be able to provide the corresponding service.

To continue the analogy, the Grid Engine software arbitrates available resources and job requirements in the following way:

- A user who submits a job through the Grid Engine software declares a requirement profile for the job. In addition, the software retrieves the identity of the user. The software also retrieves the user's affiliation with projects or user groups. The time that the user submitted the job is also stored.
- The moment that a queue is available to run a new job, the Grid Engine software determines what are the suitable jobs for the queue. The software immediately dispatches the job that has either the highest priority or the longest waiting time.
- Queues allow concurrent execution of many jobs. The Grid Engine software tries to start new jobs in the least loaded and most suitable queue.

1.4 Usage Policies

The administrator of a cluster can define high-level usage policies that are customized according to the site. Four usage policies are available:

- Urgency - Using this policy, each job's priority is based on an urgency value. The urgency value is derived from the job's resource requirements, the job's deadline specification, and how long the job waits before it is run.
- Functional - Using this policy, an administrator can provide special treatment because of a user's or a job's affiliation with a certain user group, project, and so forth.
- Share-based - Under this policy, the level of service depends on an assigned share entitlement, the corresponding shares of other users and user groups, the past usage of resources by all users, and the current presence of users within the system.
- Override - This policy requires manual intervention by the cluster administrator, who modifies the automated policy implementation.

Policy management automatically controls the use of shared resources in the cluster to best achieve the goals of the administration. High priority jobs are dispatched preferentially. Such jobs receive higher CPU entitlements if the jobs compete for resources with other jobs. The Grid Engine software monitors the progress of all jobs and adjusts their relative priorities correspondingly and with respect to the goals defined in the policies.

1.4.1 Using Tickets to Administer Policies

The functional, share-based, and override policies are defined through a Grid Engine concept that is called tickets. You might compare tickets to shares of a public company's stock. The more shares of stock that you own, the more important you are to the company. If shareholder A owns twice as many shares as shareholder B, A also has twice the votes of B. Therefore shareholder A is twice as important to the company. Similarly, the more tickets that a job has, the more important the job is. If job A has twice the tickets of job B, job A is entitled to twice the resource usage of job B.

Jobs can retrieve tickets from the functional, share-based, and override policies. The total number of tickets, as well as the number retrieved from each ticket policy, often changes over time.

The administrator controls the number of tickets that are allocated to each ticket policy in total. Just as ticket allocation does for jobs, this allocation determines the relative importance of the ticket policies among each other. Through the ticket pool that is assigned to particular ticket policies, the Grid Engine software can run in different ways. For example, the software can run in a share-based mode only. Or the software can run in a combination of modes, for example, 90% share-based and 10% functional.

1.4.2 Using the Urgency Policy to Assign Job Priority

The urgency policy can be used in combination with two other job priority specifications:

- The number of tickets assigned by the functional, share-based, and override policies
- A priority value specified by the `qsub -p` command

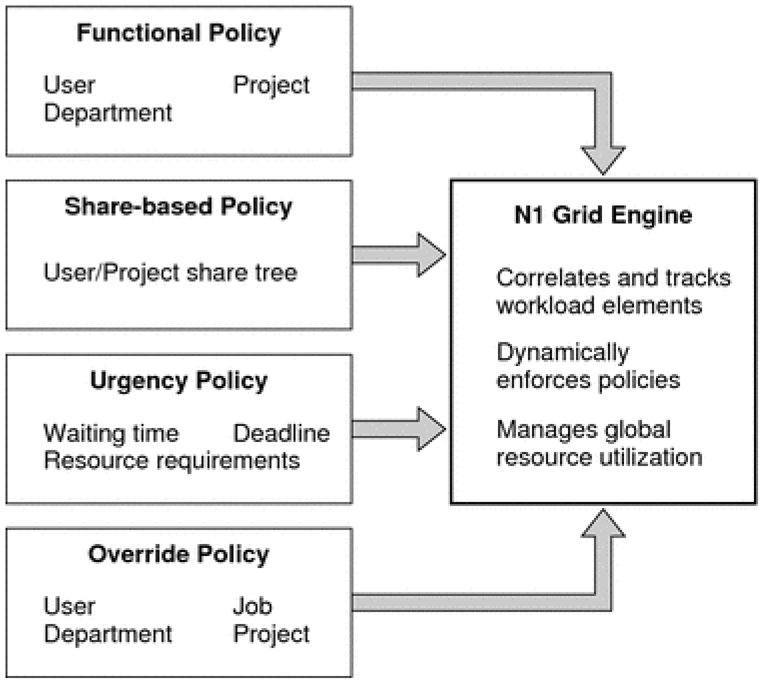
A job can be assigned an urgency value, which is derived from three sources:

- The job's resource requirements
- The length of time that a job must wait before the job runs
- The time at which a job must finish running

The administrator can separately weight the importance of each of these sources to arrive at a job's overall urgency value. For more information, see *Oracle Grid Engine Administration Guide* for managing policies.

The following figure shows the correlation among policies in a Grid Engine system.

Figure 1–2 Policy Correlation in Grid Engine



1.5 Choosing a User Interface

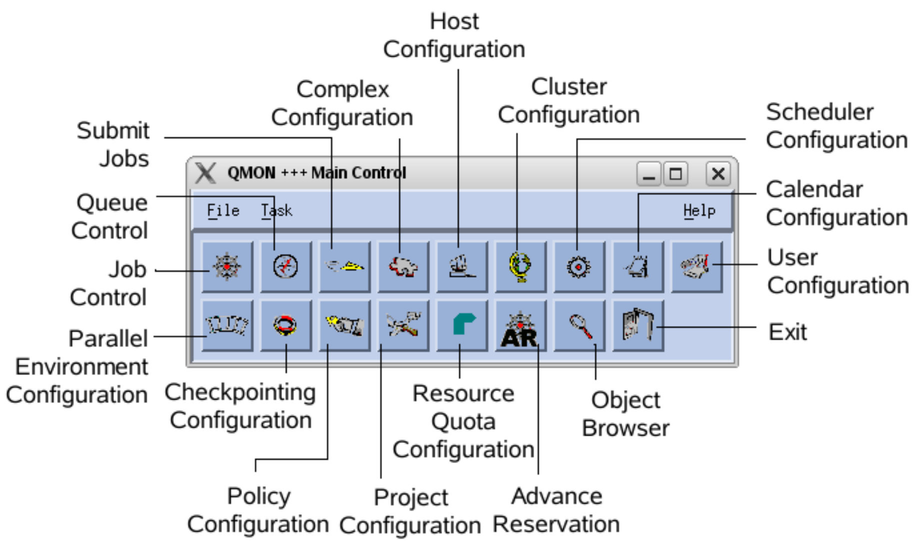
To meet the needs of your environment, the following interface tools are available:

- QMON - The Graphical User Interface
- The Command Line Interface
- The Distributed Resource Management Application API (DRMAA)

1.5.1 QMON - The Graphical User Interface

If you prefer using a graphical user interface, you can use QMON to accomplish most Grid Engine system tasks. The QMON Main Control window, which is show below, is often the starting point for user and administrator functions.

Figure 1–3 QMON Main Control Window



For more information on QMON if you are an administrator, see *Oracle Grid Engine Administration Guide* for interacting with Grid Engine as an administrator. For more information on QMON if you are an user, see Interacting With Grid Engine as a User.

1.5.2 The Command Line Interface

If you prefer using the command line, the command line user interface includes a flexible a set of ancillary programs (commands) that enable you to interact with the Grid Engine system.

For more information on the command line if you are an administrator, see *Oracle Grid Engine Administration Guide* for interacting with Grid Engine as an administrator. For more information on the command line if you are an user, see Interacting With Grid Engine as a User. For information on the ancillary programs that Grid Engine provides and which users have access to these commands, see Command Line Interface Ancillary Programs.

1.5.3 The Distributed Resource Management Application API (DRMAA)

You can automate Grid Engine functions by writing scripts that run Grid Engine commands and parse the results. However, for more consistent and efficient results, you can use the Distributed Resource Management Application API (DRMAA). For more information about the DRMAA concept and how to use it with the C and Java ™ languages, see Automating Grid Engine Functions Through DRMAA.

1.6 Users and User Categories

There are four categories of users that each have access to their own set of Grid Engine system commands:

- **Managers** - Managers have full capabilities to manipulate the Grid Engine system. By default, the superusers of all administration hosts have manager privileges.

- **Operators** - Users who can perform the same commands as managers except that they cannot change the configuration. Operators are supposed to maintain operation.
- **Users** - People who can submit jobs to the grid and run them if they have a valid login ID on at least one submit host and one execution host. Users have no cluster management or queue management capabilities.
- **Owners** - Users who can suspend or resume and disable or enable the queues they own. Typically, users are owners of the queue instances that reside on their workstations. Queue owners can be managers, operators, or users. Users are commonly declared to be owners of the queue instances that reside on their desktop workstations. See *Oracle Grid Engine Administration Guide* for more information about configuring owners parameters with QMON.

For information on which command capabilities are available to the different user categories, see Command Line Interface Ancillary Programs.

2 Using Grid Engine

This section focuses on using Grid Engine to perform tasks that distribute workload across your grid systems:

Topic	Description
Interacting With Grid Engine as a User	Learn how you can use the command line interface, the graphical user interface (QMON), and the Distributed Resource Management Application API (DRMAA) to interact with the Grid Engine system.
Displaying User Properties	Learn how to display user properties.
Displaying Host Properties	Learn how to display host properties.
Displaying Queue Properties	Learn how to display queue properties.
Submitting Jobs	Learn how to submit jobs.
Monitoring Hosts from the Command Line	Learn how to monitor and control hosts.
Monitoring and Controlling Jobs	Learn how to monitor and control jobs.
Monitoring and Controlling Queues	Learn how to monitor and control queues.
Using Job Checkpointing	Learn how to use job checkpointing as another method for monitoring jobs.
Managing Core Binding	Learn how to bind jobs to processor cores on the execution host.
Using the Accounting and Reporting Console	Learn how to gather and view information about how effectively your workload distribution uses resources.

2.1 Interacting With Grid Engine as a User

This section describes how to launch the QMON from the command line, customize the QMON, and use the command-line interface.

2.1.1 Launching QMON From the Command Line

To launch QMON from the command line, type the following command:

```
qmon
```

2.1.2 Customizing QMON

A specifically designed resource file largely defines the QMON look and feel. Reasonable defaults are compiled in `$SGE_ROOT/qmon/Qmon`. This file also includes a sample resource file. Refer to the comment lines in the sample `Qmon` file for detailed information on the possible customizations.

Users can configure the following personal preferences:

- Users can modify the `Qmon` file.
- The Qmon file can be moved to the home directory or to another location pointed to by the private `XAPPLRESDIR` search path.
- Users can include the necessary resource definitions in their private `.Xdefaults` or `.Xresources` files. A private Qmon resource file can also be installed using the `xrdb` command. The `xrdb` command can be used during operation. `xrdb` can also be used at startup of the X11 environment, for example, in a `.xinitrc` resource file.

You can also use the Job Customize and Queue Customize dialog boxes to customize QMON. These dialog boxes are shown in Monitoring and Controlling Jobs. In both dialog boxes, users can use the Save button to store the filtering and display definitions to the `.qmon_preferences` file in their home directories. When QMON is restarted, this file is read, and QMON reactivates the previously defined behavior.

For information on what your administrator can configure, see *Oracle Grid Engine Administration Guide*.

2.1.3 Using the Command-Line Interface

As a user, you will find the following commands particularly useful:

- `qalter` - Modify a pending batch job.
- `qdel` - Delete a queue.
- `qhost` - Show the status of hosts, queues, and jobs.
- `qlogin` - Submit an interactive login session.
- `qrsh` - Submit an interactive `rsh` session.
- `qsub` - Submit Jobs
- `qstat` - Check the status of a job queue.
- `qtcsh` - Used as interactive command interpreter as well as for the processing of tcsh shell scripts.

For a complete list of ancillary programs, see Command Line Interface Ancillary Programs.

2.2 Displaying User Properties

For information on the different categories of Grid Engine users, see Users and User Categories.

2.2.1 User Access Permissions

Note: The Grid Engine software automatically takes into account the access restrictions configured by the cluster administration. The following sections are important only if you want to query your personal access permission.

The administrator can restrict access to queues and other facilities, such as parallel environment interfaces. Access can also be restricted to certain users or user groups. For more information on how administrators configure access lists, see *Oracle Grid Engine Adminintration Guide* for configuring user access.

Users who belong to ACLs that are listed in access-allowed-lists have permission to access the queue or the parallel environment interface. Users who are members of ACLs in access-denied-lists cannot access the resource in question.

ACLs are also used to define projects, to which assigned users can submit their jobs. The administrator can also restrict access to cluster resources on a per project basis. For more on projects, see *Oracle Grid Engine Administration Guide* for configuring projects.

The User Configuration dialog box opens when you click the User Configuration button in the QMON Main Control window. This dialog box enables you to query for the ACLs to which you have access. For details, see *Oracle Grid Engine Administration Guide* for managing user access.

You can display project access by clicking the Project Configuration icon in the QMON Main Control window. Details are described in *Oracle Grid Engine Administration Guide* for configuring projects.

The ACLs consist of user account names and UNIX group names. The UNIX group names are identified by a prefixed `@` sign. In this way, you can determine which ACLs your account belongs to.

Note: If you have permission to switch your primary UNIX group with the `newgrp` command, your access permissions might change.

You can check for those queues or parallel environment interfaces to which you have access or to which your access is denied. Query the queue or parallel environment interface configuration, as described in Displaying Queue Properties and *Oracle Grid Engine Administration Guide* for managing parallel environments.

The access-allowed-lists are named `user_lists`. The access-denied-lists are named `xuser_lists`. If your user account or primary UNIX group is associated with an access-allowed-list, you are allowed to access the resource in question. If you are associated with an access-denied-list, you cannot access the queue or parallel environment interface. If both lists are empty, every user with a valid account can access the resource in question.

If you have access to a project, you are allowed to submit jobs that are subordinated to the project. You can submit such jobs from the command line using the following command:

```
% qsub -P <project-name> <options>
```

The cluster configurations, host configurations, and queue configurations define project access in the same way as for ACLs. These configurations use the `project_lists` and `xproject_lists` parameters for this purpose.

2.2.2 Displaying Managers, Operators, Owners, and User Access Permissions

Note: The superuser of an administration host is considered to be a manager by default.

How to Display A List of Managers From the Command Line

To display a list of managers, type the following command:

```
qconf -sm
```

How to Display a List of Managers With QMON

1. Click on the User Configuration button on the QMON Main Control window.
2. Click on the Manager tab.

   A list of currently-configured managers are displayed.

How to Display A List of Operators From the Command Line

To display a list of operators, type the following command:

```
qconf -so
```

How to Display a List of Operators With QMON

1. Click on the User Configuration button on the QMON Main Control window.
2. Click on the Operator tab. A list of currently-configured operators are displayed.

How to Display a List of Owners From the Command Line

To display a list of owners, type the following command:

```
qconf -sq {<cluster-queue> | <queue-instance> | <queue-domain>}
```

How to Display a List of Owners With QMON

1. Click on the User Configuration button on the QMON Main Control window.
2. Click on the Owner tab.

How to Display User Access Lists From the Command Line

To display a list of currently configured ACLS, type the following command:

```
qconf -sul
```

To display a list of currently configured ACLS, type the following command:

```
qconf -su <acl-name> [,<...>]
```

How to Display User Access Lists With QMON

1. Click User Configuration on the QMON Main Control window.
2. Click the Userset tab. This dialog box enables you to query for the ACLs to which you have access. You can also see what projects to which you have access. For more on projects, see *Oracle Grid Engine Administration Guide*.

How to Display a List of Defined Projects From the Command Line

To display a list of all defined projects, type the following command:

```
qconf -sprjl
```

To display a specific project configuration, type the following command:

```
qconf -sprj <project-name>
```

How to Display a List of Defined Projects With QMON

2.3 Displaying Host Properties

Clicking the Host Configuration button in the QMON Main Control window displays an overview of the functionality that is associated with the hosts in your cluster. You need to have manager privileges to apply any changes to the configuration.

The host configuration dialog boxes are described in *Oracle Grid Engine Administration Guide* for configuring hosts.

How to Display the Name of the Master Host From the Command Line

The location of the master host can migrate between the current master host and one of the shadow master hosts at any time. Therefore, the location of the master host should be transparent to the user.

To display the name of the master host, view `$SGE_ROOT/$SGE_CELL/common/act_qmaster` file in a text editor.

The name of the current master host is listed in the file.

How to Display a List of Execution Hosts From the Command Line

To display a complete list of the execution hosts in your cluster, type the following command:

```
qconf -sel
```

To display the configuration for a specific execution host, type the following command:

```
qconf -se <hostname>
```

To display status and load information about execution hosts, type the following command:

```
qhost
```

How to Display a List of Administration Host From the Command Line

To display a list of administration hosts, type the following command:

```
qconf -sh
```

How to Display a List of Submit Hosts From the Command Line

To display a list of submit hosts, type the following command:

```
qconf -ss
```

2.4 Displaying Queue Properties

To make the best use of the Grid Engine system at your site, you should be familiar with the queue structure. You should also be familiar with the properties of the queues that are configured for your Grid Engine system.

How to Display a List of Queues From the Command Line

To display a list of queues from the command line, type the following command:

```
% qconf -sql
```

How to Display a List of Queues With QMON

1. Launch the QMON Main Control window.
2. Click the Queue Control button. The Cluster Queue Control dialog box appears. Queue Control dialog box provides a quick overview of the installed queues and their current status.

How to Display Queue Properties From the Command Line

To display queue properties from the command line, type the following command:

```
% qconf -sq {<queue> | <queue-instance> | <queue-domain>}
```

How to Display Queue Properties With QMON

1. Launch the QMON Main Control window.
2. Click the Queue Control button. The Cluster Queue Control dialog box appears.
3. Select a queue, and then click Show Detached Settings. The Browser dialog box appears.
4. In the Browser dialog box, click Queue.
5. In the Cluster Queue dialog box, click the Queue Instances tab.
6. Select a queue instance. The Browser dialog box lists the queue properties for the selected queue instance.

2.4.1 Interpreting Queue Property Information

The following is a list of some of the more important parameters:

- `qname` - The queue name as requested.
- `hostlist` - A list of hosts and host groups associated with the queue.
- `processors` - The processors of a multiprocessor system to which the queue has access.

  Caution: Do not change this value unless you are certain that you need to change it.

- `qtype` - The type of job that can run in this queue. Currently, the type can be either batch or interactive.
- `slots` - The number of jobs that can be executed concurrently in that queue.
- `owner_list` - The owners of the queue. For more information, see Users and User Categories.

- `user_lists` - The user or group identifiers in the user access lists who can access the queue. For more information, see Displaying User Properties.
- `xuser_lists` - The user or group identifiers in the user access lists who cannot access the queue. For more information, see Displaying User Properties.
- `project_lists` - The jobs submitted with the project identifiers that can access the queue. For more information, see *Oracle Grid Engine Administration Guide*.
- `xproject_lists` - The jobs submitted with the project identifiers that cannot access the queue. For more information, see *Oracle Grid Engine Administration Guide*.
- `complex_values` - Assigns capacities as provided for this queue for certain complex resource attributes. For more information, see Requestable Attributes.

2.5 Submitting Jobs

A job is a segment of work. Each job includes a description of what to do and a set of property definitions that describe how the job should be run.

The Grid Engine system recognizes the following four basic classes of jobs:

- Batch Jobs - Single segments of work. Typically, a batch job is only executed once.
- Array Jobs - Groups of similar work segments that can all be run in parallel but are completely independent of one another. All of the workload segments of an array job, known as tasks, are identical except for the data sets on which they operate.
- Parallel Jobs - Jobs composed of cooperating tasks that must all be executed at the same time, often with requirements about how the tasks are distributed across the resources.
- Interactive Jobs - Jobs that provide the submitting user with an interactive login to an available resource in the compute cluster. Interactive jobs allow users to execute work on the compute cluster that is not easily submitted as a batch job.

2.5.1 How Jobs Are Scheduled

The Grid Engine system schedules jobs using the following process:

1. A scheduling run is triggered in one of the following ways:
   - At a fixed interval. The default is every 15 seconds.
   - By new job submissions or notification from an execution daemon that one or more jobs has finished executing.
   - By using `qconf -tsm`, which an administrator can use to trigger a scheduling run.
2. The scheduler assesses the needs of all pending jobs against available resources by considering the following:

   Note: If share-based scheduling is used, the calculation takes into account the usage that has already occurred for that user or project.

   - Administrator's specifications for jobs and queues
   - Each pending job's resource requirements (for example, CPU, memory, and I/O bandwidth)

- Resource reservations that need to be made for future jobs
- The cluster's current load
- The host's relative performance

3. As a result of the scheduler's assessment, the Grid Engine system does the following tasks, as needed:
   - Dispatches new jobs
   - Suspends running jobs
   - Increases or decreases the resources allocated to running jobs
   - Maintains the status quo

Between scheduling actions, the Grid Engine system keeps information about significant events such as the following:

- Job submission
- Job completion
- Job cancellation
- An update of the cluster configuration
- Registration of a new machine in the cluster

2.5.2 Usage Policies

The Grid Engine software's policy management automatically controls the use of shared resources in the cluster to best achieve the goals of the administration. High priority jobs are dispatched preferentially and receive better access to resources.

The cluster administrator can define high-level usage policies. The following policies are available:

- Functional - Special treatment is given because of affiliation with a certain user group, project, and so forth.
- Share-based - Level of service depends on an assigned share entitlement, the corresponding shares of other users and user groups, the past usage of resources by all users, and the current presence of users in the system.
- Urgency - Preferential treatment is given to jobs that have greater urgency. A job's urgency is based on its resource requirements, how long the job must wait, and whether the job is submitted with a deadline requirement.
- Override - Manual intervention by the cluster administrator modifies the automated policy implementation.

The Grid Engine software can be set up to routinely use either a share-based policy, a functional policy, or both. These policies can be combined in any proportion, from giving zero weight to one policy and using only the second policy, to giving both policies equal weight. Administrators can temporarily override share-based scheduling and functional scheduling. An override can be applied to an individual job or to all jobs associated with a user, a department, or a project. For more information, see *Oracle Grid Engine Adminsitration Guide* for managing policies.

Along with the routine policies, jobs can be submitted with an initiation deadline. See the description of the deadline submission parameter under How to Submit an Advanced Job With QMON. Deadline jobs disturb routine scheduling.

2.5.3 Job Priorities

The Grid Engine software also lets users set individual job priorities. A user who submits several jobs can specify, for example, that job 3 is the most important and that jobs 1 and 2 are equally important but less important than job 3.

Use one of the following options to set priorities:

- QMON Submit Job parameter Priority
- `qsub -p` option.

You can set a priority range of -1023 (lowest) to 1024 (highest). This priority tells the scheduler how to choose among users' jobs when several jobs are in the system simultaneously.

Note: Since users are not permitted to submit jobs with a priority higher than 0, which is the default, a best administrative practice is to set the default priority at a lower priority, that is, `-100`.

2.5.4 Ticket Policies

The functional policy, the share-based policy, and the override policy are all implemented with tickets. Each ticket policy has a ticket pool from which tickets are allocated to jobs that are entering the Grid Engine system. Each routine ticket policy that is in force allocates some tickets to each new job. The ticket policy can reallocate tickets to the executing job at each scheduling interval.

Tickets weight the three ticket policies. For example, if no tickets are allocated to the functional policy, then that policy is not used. If an equal number of tickets are assigned to the functional ticket pool and to the share-based ticket pool, then both policies have equal weight in determining a job's importance.

The following are criteria that each ticket policy uses to allocate tickets:

- Grid Engine managers allocate tickets to the routine ticket policies at system configuration. Managers and operators can change ticket allocations at any time. Additional tickets can be injected into the system temporarily to indicate an override. Ticket policies can be combined when tickets are allocated to multiple ticket policies, a job gets a portion of its tickets from each ticket policy.
- The Grid Engine system grants tickets to jobs that are entering the system to indicate their importance under each ticket policy. Each running job can gain tickets, for example, from an override; lose tickets, for example, because the job is getting more than its fair share of resources; or keep the same number of tickets at each scheduling interval. The number of tickets that a job holds represents the resource share that the Grid Engine system tries to grant that job during each scheduling interval.

You can display the number of tickets a job holds with QMON or using `qstat -ext`. See How to Monitor Jobs With QMON. The `qstat` command also displays the priority value assigned to a job, for example, using `qsub -p`.

2.5.5 Queue Selection

Jobs that are submitted to a named queue go directly to the named queue, regardless of whether the jobs can be started or need to be spooled. Jobs that are not submitted to a named queue that cannot be started immediately are put into a spool. The `sge_qmaster` then tries to reschedule the jobs until a suitable queue becomes available, allowing the jobs to be dispatched. Therefore, viewing the queues of the Grid Engine system as computer science batch queues is valid only for jobs requested by name. Jobs submitted with nonspecific requests use the spooling mechanism of `sge_qmaster` for queueing, thus using a more abstract and flexible queuing concept.

If a job is scheduled and multiple free queues meet its resource requests, the job is usually dispatched to a suitable queue belonging to the least loaded host. By setting the scheduler configuration entry `queue_sort_method` to `seq_no`, the cluster administration can change this load-dependent scheme into a fixed order algorithm. The queue configuration entry `seq_no` defines a precedence among the queues, assigning the highest priority to the queue with the lowest sequence number.

2.5.6 Defining Resource Requirements

In the examples so far, the submit options do not express any resource requirements for the hosts on which the jobs are to be executed. The Grid Engine system assumes that such jobs can be run on any host. In practice, however, most jobs require that certain prerequisites be met on the executing host in order for the job to finish successfully. These prerequisites include:

- Enough available memory
- Installation of required software
- Certain operating system architecture

Also, the cluster administrator usually imposes restrictions on the use of the machines in the cluster. For example, the CPU time that can be consumed by the jobs is often restricted.

The Grid Engine system provides users with the means to find suitable hosts for their jobs without precise knowledge of the cluster`s equipment and its usage policies. Users specify the requirement of their jobs and let the Grid Engine system manage the task of finding a suitable and lightly loaded host.

You specify resource requirements through requestable attributes, which are described in Requestable Attributes. QMON provides a convenient way to specify the requirements of a job. The Requested Resources dialog box displays only those attributes in the Available Resource list that are currently eligible. Click Request Resources in the Submit Job dialog box to open the Requested Resources dialog box.

When you double-click an attribute, the attribute is added to the Hard or Soft Resources list of the job. A dialog box opens to guide you in entering a value specification for the attribute in question, except for BOOLEAN attributes, which are set to True. For more information, see How the Grid Engine System Allocates Resources.

Figure 2–1 shows a resource profile for a job that requests a `solaris64` host with an available `permas` license offering at least 750 MBytes of memory. If more than one queue that fulfills this specification is found, any defined soft resource requirements are taken into account. However, if no queue satisfying both the hard and the soft requirements is found, any queue that grants the hard requirements is considered suitable.

Note: The `queue_sort_method` parameter of the scheduler configuration determines where to start the job only if more than one queue is suitable for a job.

The attribute `permas`, an integer, is an administrator extension to the global resource attributes. The attribute arch, a string, is a host resource attribute. The attribute `h_vmem`, memory, is a queue resource attribute.

An equivalent resource requirement profile can as well be submitted from the `qsub` command line:

```
% qsub -l arch=solaris64,h_vmem=750M,permas=1 \
    permas.sh
```

The implicit `-hard` switch before the first `-l` option has been skipped.

The notation `750M` for 750 MBytes is an example of the quantity syntax of the Grid Engine system. For those attributes that request a memory consumption, you can specify either integer decimal, floating-point decimal, integer octal, and integer hexadecimal numbers. The following multipliers must be appended to these numbers:

- k - Multiplies the value by 1000
- K - Multiplies the value by 1024
- m - Multiplies the value by 1000 times 1000
- M - Multiplies the value by 1024 times 1024

Octal constants are specified by a leading zero and digits ranging from 0 to 7 only. To specify a hexadecimal constant, you must prefix the number with 0x. You must also use digits ranging from 0 to 9, a through f, and A through F. If no multipliers are appended, the values are considered to count as bytes. If you are using floating-point decimals, the resulting value is truncated to an integer value.

For those attributes that impose a time limit, you can specify time values in terms of hours, minutes, or seconds, or any combination. Hours, minutes, and seconds are specified in decimal digits separated by colons. A time of 3:5:11 is translated to 11111 seconds. If zero is a specifier for hours, minutes, or seconds, you can leave it out if the colon remains. Thus a value of :5: is interpreted as 5 minutes. The form used in the Requested Resources dialog box that is shown in Figure 2–1 is an extension, which is valid only within QMON.

How the Grid Engine System Allocates Resources

Knowing how the Grid Engine software processes resource requests and allocates resources is important. The resource allocation algorithm that Grid Engine software uses is as follows:

1. Read in and parse all default request files. See Default Request Files for details.
2. Process the script file for embedded options. See Active Comments for details.
3. Read all script-embedding options when the job is submitted, regardless of their position in the script file.
4. Read and parse all requests from the command line.

As soon as all `qsub` requests are collected, hard and soft requests are processed separately.

The requests are evaluated in the following order of precedence:

1. From left to right of the script or default request file.
2. From top to bottom of the script or default request file.
3. From left to right of the command line. In other words, you can use the command line to override the embedded flags.

Hard requests are processed first. If a hard request is not valid, the submission is rejected. If one or hard more requests cannot be met at submit time, the job is spooled and rescheduled to be run at a later time. For example, a hard request might not be met if a requested queue is busy. If all hard requests can be met, the resources are allocated and the job can be run.

The soft resource requests are then checked. The job can run even if some or all of these requests cannot be met. If multiple queues that meet the hard requests provide parts of the soft resources list, the Grid Engine software selects the queues that offer the most soft requests.

The job is started and covers the allocated resources.

You might want to gather experience of how argument list options and embedded options or hard and soft requests influence each other. You can experiment with small test script files that execute UNIX commands such as hostname or date.

2.5.7 Requestable Attributes

When you submit a job, a requirement profile can be specified. You can specify attributes or characteristics of a host or queue that the job requires to run successfully.

The attributes that can be used to specify the job requirements are related to one of the following:

- The cluster, for example, space required on a network shared disk
- Individual hosts, for example, operating system architecture
- Queues, for example, permitted CPU time

The attributes can also be derived from site policies such as the availability of installed software only on certain hosts.

The available attributes include the following:

- Queue property list - See Displaying Queue Properties.
- List of global and host-related attributes - See *Oracle Grid Engine Administration Guide* for more information about assigning resource attributes to queues, hosts, and the global cluster.
- Administrator-defined attributes

For convenience, however, the administrator commonly chooses to define only a subset of all available attributes to be requestable.

The Grid Engine system complex contains the definitions for all resource attributes. For more information about resource attributes, see *Oracle Grid Engine Administration Guide* for configuring resource attributes.

How to Display Requestable Attributes From the Command Line

From the command line, type the following:

```
% qconf -sc
```

The following example shows sample output from the `qconf -sc` command:

```
gimli% qconf -sc
#name              shortcut   type        relop requestable consumable default
urgency
#----------------------------------------------------------------------------------
-------
arch               a          RESTRING    ==    YES         NO         NONE     0
```

```
calendar            c          STRING      ==    YES         NO         NONE     0
cpu                 cpu        DOUBLE      >=    YES         NO         0        0
h_core              h_core     MEMORY      <=    YES         NO         0        0
h_cpu               h_cpu      TIME        <=    YES         NO         0:0:0    0
h_data              h_data     MEMORY      <=    YES         NO         0        0
h_fsize             h_fsize    MEMORY      <=    YES         NO         0        0
h_rss               h_rss      MEMORY      <=    YES         NO         0        0
h_rt                h_rt       TIME        <=    YES         NO         0:0:0    0
h_stack             h_stack    MEMORY      <=    YES         NO         0        0
h_vmem              h_vmem     MEMORY      <=    YES         NO         0        0
hostname            h          HOST        ==    YES         NO         NONE     0
load_avg            la         DOUBLE      >=    NO          NO         0        0
load_long           ll         DOUBLE      >=    NO          NO         0        0
load_medium         lm         DOUBLE      >=    NO          NO         0        0
load_short          ls         DOUBLE      >=    NO          NO         0        0
mem_free            mf         MEMORY      <=    YES         NO         0        0
mem_total           mt         MEMORY      <=    YES         NO         0        0
mem_used            mu         MEMORY      >=    YES         NO         0        0
min_cpu_interval    mci        TIME        <=    NO          NO         0:0:0    0
np_load_avg         nla        DOUBLE      >=    NO          NO         0        0
np_load_long        nll        DOUBLE      >=    NO          NO         0        0
np_load_medium      nlm        DOUBLE      >=    NO          NO         0        0
np_load_short       nls        DOUBLE      >=    NO          NO         0        0
num_proc            p          INT         ==    YES         NO         0        0
qname               q          STRING      ==    YES         NO         NONE     0
rerun               re         BOOL        ==    NO          NO         0        0
s_core              s_core     MEMORY      <=    YES         NO         0        0
s_cpu               s_cpu      TIME        <=    YES         NO         0:0:0    0
s_data              s_data     MEMORY      <=    YES         NO         0        0
s_fsize             s_fsize    MEMORY      <=    YES         NO         0        0
s_rss               s_rss      MEMORY      <=    YES         NO         0        0
s_rt                s_rt       TIME        <=    YES         NO         0:0:0    0
s_stack             s_stack    MEMORY      <=    YES         NO         0        0
s_vmem              s_vmem     MEMORY      <=    YES         NO         0        0
seq_no              seq        INT         ==    NO          NO         0        0
slots               s          INT         <=    YES         YES        1
1000
swap_free           sf         MEMORY      <=    YES         NO         0        0
swap_rate           sr         MEMORY      >=    YES         NO         0        0
swap_rsvd           srsv       MEMORY      >=    YES         NO         0        0
swap_total          st         MEMORY      <=    YES         NO         0        0
swap_used           su         MEMORY      >=    YES         NO         0        0
tmpdir              tmp        STRING      ==    NO          NO         NONE     0
virtual_free        vf         MEMORY      <=    YES         NO         0        0
virtual_total       vt         MEMORY      <=    YES         NO         0        0
virtual_used        vu         MEMORY      >=    YES         NO         0        0
# >#< starts a comment but comments are not saved across edits --------
```

The column name is identical to the first column displayed by the `qconf -sq` command. The shortcut column contains administrator-definable abbreviations for the full names in the first column. The user can supply either the full name or the shortcut in the request option of a `qsub` command.

The column requestable tells whether the resource attribute can be used in a `qsub` command. The administrator can, for example, disallow the cluster's users to request certain machines or queues for their jobs directly. The administrator can disallow direct requests by setting the entries qname, hostname, or both, to be unrequestable. Making queues or hosts unrequestable implies that feasible user requests can be met in general by multiple queues, which enforces the load balancing capabilities of the Grid Engine system.

The column relop defines the relational operator used to compute whether a queue or a host meets a user request. The comparison that is executed is as follows:

```
User_Request     relop     Queue/Host/... -Property
```

If the result of the comparison is false, the user's job cannot be run in the queue or on the host. For example, let the queue `q1` be configured with a soft CPU time limit of 100 seconds. Let the queue `q2` be configured to provide 1000 seconds soft CPU time limit.

The columns consumable and default affect how the administrator declares consumable resources. See *Oracle Grid Engine Administration Guide* for consumable resources.

The user requests consumables just like any other attribute. The Grid Engine system internal bookkeeping for the resources is different, however.

Assume that a user submits the following request:

```
% qsub -l s_cpu=0:5:0 nastran.sh
```

The `s_cpu=0:5:0` request asks for a queue that grants at least 5 minutes of soft limit CPU time. Therefore, only queues providing at least 5 minutes soft CPU runtime limit are set up properly to run the job.

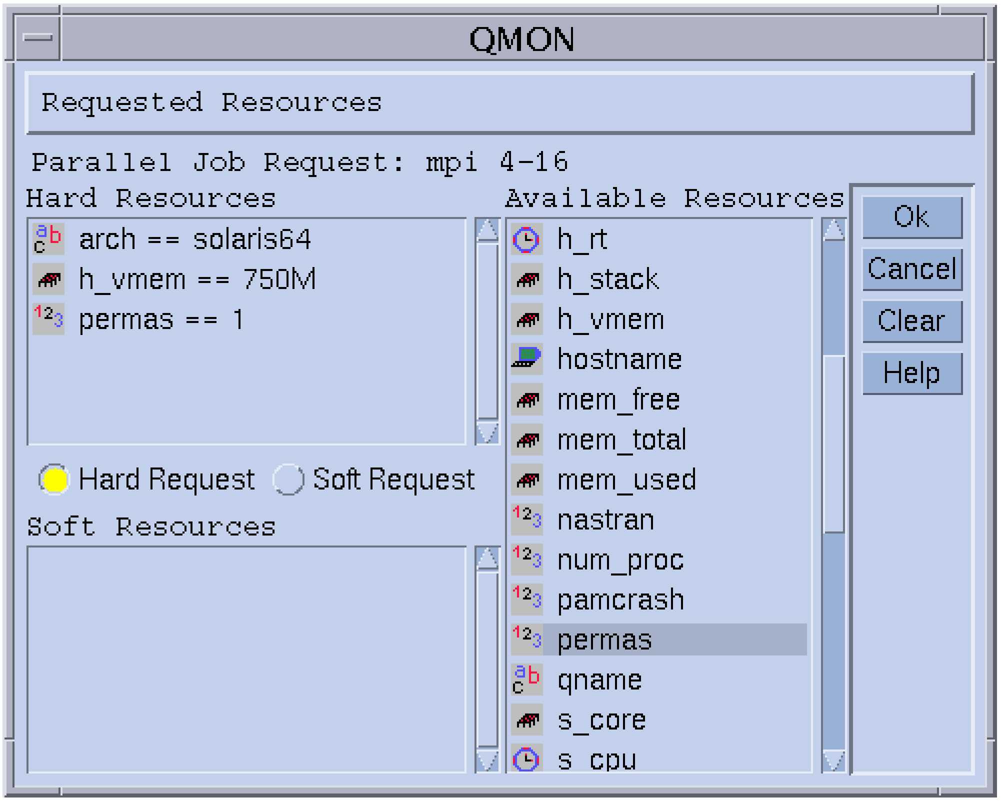
For boolean complex values, and for complexes of type STRING and CSTRING, the value TRUE is the default and will be used if no explicit value is specified. For integer-based complex values, the value 1 is the default and will be used if no explicit value is specified.

Note: The Grid Engine software considers workload information in the scheduling process only if more than one queue or host can run a job.

How to Display Requestable Attributes With QMON

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select a pending job and click the Submit button. The Submit Job dialog box appears.
3. Click the Request Resources button. The Requested Resources dialog box displays the currently requestable attributes under Available Resources, which is shown in the following figure.

Figure 2–1 Request Resources Window


QMON

Requested Resources

Parallel Job Request: mpi 4-16

Hard Resources

arch == solaris64
h_vmem == 750M
permas == 1

Hard Request [✓] Soft Request

Soft Resources

Available Resources

h_rt
h_stack
h_vmem
hostname
mem_free
mem_total
mem_used
nastran
num_proc
pamcrash
permas
qname
s_core
s_cpu

2.6 Submitting Batch Jobs

The following sections describe how to submit more complex jobs through the Grid Engine system.

For information about submitting simple jobs, see Submitting Jobs.

2.6.1 About Shell Scripts

Shell scripts, also called batch jobs, are a sequence of command-line instructions that are assembled in a file. Each instruction is interpreted as if the instruction were typed manually by the user who is running the script. You can invoke arbitrary commands, applications, and other shell scripts from within a shell script.

Script files are made executable by the `chmod` command. If scripts are invoked, a command interpreter is started. `csh`, `tcsh`, `sh`, or `ksh` are typical command interpreters.

The command interpreter can be invoked as login shell. To do so, the name of the command interpreter must be contained in the `login_shells` list of the Grid Engine system configuration that is in effect for the particular host and queue that is running the job.

Note: The Grid Engine system configuration might be different for the various hosts and queues configured in your cluster. You can display the effective configurations with the `-sconf` and `-sq` options of the `qconf` command.

If the command interpreter is invoked as login shell, your job environment is the same as if you logged in and ran the script. In using csh, for example, .login and .cshrc are executed in addition to the system default startup resource files, such as /etc/login, whereas only .cshrc is executed if csh is not invoked as login-shell. For a description of the difference between being invoked and not being invoked as login-shell, see the man page for your command interpreter.

Example of a Shell Script

The following example is a simple shell script that compiles the application flow from its Fortran77 source and then runs the application:

```
#!/bin/csh
# This is a sample script file for compiling and
# running a sample FORTRAN program under N1 Grid Engine 6
cd TEST
# Now we need to compile the program "flow.f" and
# name the executable "flow".
f77 flow.f -o flow
```

Your local system user's guide provides detailed information about building and customizing shell scripts. You might also want to look at the sh, ksh, csh, or tcsh man pages. The following sections emphasize special things that you should consider when you prepare batch scripts for the Grid Engine system.

In general, you can submit all shell scripts to the Grid Engine system that you can run from your command prompt by hand. These shell scripts must not require a terminal connection or need interactive user intervention. The exceptions are the standard error and standard output devices, which are automatically redirected.

2.6.2 Extensions to Regular Shell Scripts

Some extensions to regular shell scripts influence the behavior of scripts that run under Grid Engine system control. The following sections describe these extensions.

2.6.2.1 How a Command Interpreter is Selected

At submit time, you can specify the command interpreter to use for the job script file as shown in Figure 2–8. However, if nothing is specified, the configuration variable shell_start_mode determines how the command interpreter is selected:

- If shell_start_mode is set to unix_behavior, the first line of the script file specifies the command interpreter. The first line of the script file must begin with a pound symbol (#) followed by an exclamation point (!). If the first line does not begin with those characters, the Bourne Shell sh is used by default.
- For all other settings of shell_start_mode, the default command interpreter is determined by the shell parameter for the queue where the job starts. See Displaying Queue Properties.

2.6.2.2 Output Redirection

Since batch jobs do not have a terminal connection, their standard output and their standard error output must be redirected into files. The Grid Engine system enables the user to define the location of the files to which the output is redirected. Defaults are used if no output files are specified.

The standard location for the files is in the current working directory where the jobs run. The default standard output file name is job-name.ojob-id. The default standard error output is redirected to job-name>.ejob-id. The job_name can be built from the script file name, or defined by the user. `job-id` is a unique identifier that is assigned to the job by the Grid Engine system.

For array job tasks, the task identifier is added to these filenames, separated by a dot. The resulting standard redirection paths are `job-name.ojob-id.task-id`> and `job-name.ejob-id.task-id`. For more information, see Submitting Array Jobs.

If the standard locations are not suitable, you can use one of the following to specify output directions:

- QMON as shown in Figure 2–9
- `-e` and `-o` options to the `qsub` command

Standard output and standard error output can be merged into one file. The redirections can be specified on a per execution host basis, in which case, the location of the output redirection file depends on the host on which the job is executed. To build custom but unique redirection file paths, use dummy environment variables together with the `qsub -e` and `-o` options. A list of these variables follows:

- `HOME` - Home directory on execution machine
- `USER` - User ID of job owner
- `JOB_ID` - Current job ID
- `JOB_NAME` - Current job name; see the -N option
- `HOSTNAME` - Name of the execution host
- `TASK_ID` - Array job task index number

When the job runs, these variables are expanded into the actual values, and the redirection path is built with these values.

Note: The `qsub` job can be run in a pseudo terminal using the `-pty yes` option. If no pseudo terminal is available, the job fails. By default, `qsub` starts a job without a pseudo terminal. The `-pty no` option will force `qsub` to run without a pseudo terminal.

2.6.2.3 Active Comments

Lines with a leading # sign are treated as comments in shell scripts. The Grid Engine system also recognizes special comment lines that supply options to commands or to the QMON interface. By default, these special comment lines are identified by the #$ prefix string. You can redefine the prefix string with the `qsub -C` command.

This use of special comments is referred to as "script embedding of submit arguments." The following example shows a script file that uses script-embedded command-line options to supply arguments to the `qsub` command. These options also apply to the QMON Submit Job dialog box. The corresponding parameters are preset when a script file is selected.

Example - Using Script-Embedded Command Line Options

```
#!/bin/csh

#Force csh if not Grid Engine default
#shell

#$ -S /bin/csh
```

```
# This is a sample script file for compiling and
# running a sample FORTRAN program under N1 Grid Engine 6
# We want Grid Engine to send mail
# when the job begins
# and when it ends.

#$ -M EmailAddress
#$ -m b e

# We want to name the file for the standard output
# and standard error.

#$ -o flow.out -j y

# Change to the directory where the files are located.

cd TEST

# Now we need to compile the program "flow.f" and
# name the executable "flow".

f77 flow.f -o flow

# Once it is compiled, we can run the program.

flow
```

2.6.2.4 Environment Variables

Note: If you would to change the predefined values of these variables, use the `-V` or `-v` options with `qsub` or `qalter`.

When a job runs, the following variables are preset into the job's environment:

- `ARC` - The architecture name of the node on which the job is running. The name is compiled into the `sge_execd` binary.
- `$SGE_ROOT` - The root directory of the Grid Engine system as set for `sge_execd` before startup, or the default `/usr/SGE` directory.
- `SGE_BINARY_PATH` - The directory in which the Grid Engine system binaries are installed.
- `$SGE_CELL` - The cell in which the job runs.
- `SGE_JOB_SPOOL_DIR` - The directory used by `sge_shepherd` to store job-related data while the job runs.
- `SGE_O_HOME` - The path to the home directory of the job owner on the host from which the job was submitted.
- `SGE_O_HOST` - The host from which the job was submitted.
- `SGE_O_LOGNAME` - The login name of the job owner on the host from which the job was submitted.
- `SGE_O_MAIL` - The content of the MAIL environment variable in the context of the job submission command.

- `SGE_O_PATH` - The content of the PATH environment variable in the context of the job submission command.
- `SGE_O_SHELL` - The content of the SHELL environment variable in the context of the job submission command.
- `SGE_O_TZ` - The content of the TZ environment variable in the context of the job submission command.
- `SGE_O_WORKDIR` - The working directory of the job submission command.
- `SGE_CKPT_ENV` - The checkpointing environment under which a checkpointing job runs. The checkpointing environment is selected with the `qsub -ckpt` command.
- `SGE_CKPT_DIR` - The path `ckpt_dir` of the checkpoint interface. Set only for checkpointing jobs.
- `SGE_STDERR_PATH` - The path name of the file to which the standard error stream of the job is diverted. This file is commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.
- `SGE_STDOUT_PATH` - The path name of the file to which the standard output stream of the job is diverted. This file is commonly used for enhancing the output with messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.
- `SGE_TASK_ID` - The unique index number for an array job task. You can use the `SGE_TASK_ID` to reference various input data records. This environment variable is set to undefined for non-array jobs. It is possible to change the predefined value of this variable with the `-v` or `-V` submit option.
- `SGE_TASK_FIRST` - The index number of the first array job task. For more information, see the `-t` option for `qsub`. It is possible to change the predefined value of this variable with the `-v` or `-V` submit option.
- `SGE_TASK_LAST` - The index number of the last array job task. For more information, see the `-t` option for `qsub`. It is possible to change the predefined value of this variable with the `-v` or `-V` submit option.
- `SGE_TASK_STEPSIZE` - The step size of the array job specification. For more information, see the `-t` option for `qsub`. It is possible to change the predefined value of this variable with the `-v` or `-V` submit option.
- `ENVIRONMENT` - Always set to BATCH. This variable indicates that the script is run in batch mode.
- `HOME` - The user's home directory path as taken from the `passwd` file.
- `HOSTNAME` - The host name of the node on which the job is running.
- `JOB_ID` - A unique identifier assigned by the `sge_qmaster` daemon when the job was submitted. The job ID is a decimal integer from 1 through 9,999,999.
- `JOB_NAME` - The job name, which is built from the file name provided with the `qsub` command, a period, and the digits of the job ID. You can override this default with `qsub -N`.
- `LOGNAME` - The user's login name as taken from the `passwd` file.
- `NHOSTS`-- The number of hosts in use by a parallel job.
- `NQUEUES` - The number of queues that are allocated for the job. This number is always 1 for serial jobs.

- `NSLOTS` - The number of queue slots in use by a parallel job.
- `PATH` - A default shell search path of: `/usr/local/bin:/usr/ucb:/bin:/usr/bin`.
- `PE` - The parallel environment under which the job runs. This variable is for parallel jobs only.
- `PE_HOSTFILE` - The path of a file that contains the definition of the virtual parallel machine that is assigned to a parallel job by the Grid Engine system. This variable is used for parallel jobs only. See the description of the `$pe_hostfile` parameter in `sge_pe` for details on the format of this file.
- `QUEUE` - The name of the queue in which the job is running.
- `REQUEST` - The request name of the job. The name is either the job script file name or is explicitly assigned to the job by the `qsub -N` command.
- `RESTARTED` - Indicates whether a checkpointing job was restarted. If set to value 1, the job was interrupted at least once. The job is therefore restarted.
- `SHELL` - The user's login shell as taken from the `passwd` file.

Note: SHELL is not necessarily the shell that is used for the job.

- `TMPDIR` - The absolute path to the job's temporary working directory.
- `TMP` - The same as TMPDIR. This variable is provided for compatibility with NQS.
- `TZ` - The time zone variable imported from `sge_execd`, if set.
- `USER` - The user's login name as taken from the `passwd` file.

2.7 Submitting Array Jobs

Parameterized and repeated execution of the same set of operations that are contained in a job script is an ideal application for the array job facility of the Grid Engine system. Typical examples of such applications are found in the Digital Content Creation industries for tasks such as rendering. Computation of an animation is split into frames. The same rendering computation can be performed for each frame independently.

The Grid Engine system provides an efficient implementation of array jobs, handling the computations as an array of independent tasks joined into a single job. The tasks of an array job are referenced through an array index number. The indexes for all tasks span an index range for the entire array job. The index range is defined during submission of the array job by a single `qsub` command.

You can monitor and control an array job. For example, you can suspend, resume, or cancel an array job as a whole or by individual task or subset of tasks. To reference the tasks, the corresponding index numbers are suffixed to the job ID. Tasks are executed very much like regular jobs. Tasks can use the environment variable `SGE_TASK_ID` to retrieve its own task index number and to access input data sets designated for this task identifier.

2.7.1 How to Configure Array Task Dependencies From the Command Line

While most interdependent tasks can be supported by Grid Engine's job dependency facility, certain array jobs require the flexibility provided by the array task dependency facility. The array task dependency facility allows users to make one array job's tasks dependent on the tasks of another array job. For example, if you use Grid Engine to render video effects, the array task dependency allows you to submit each step as an array job where each task represents a frame. Each task then depends on the corresponding task in the previous step.

To configure an array task dependency, use the following command:

```
qsub -hold_jid_ad wc_job_list
```

The `-hold_jid_ad` option defines or redefines the job array dependency list of the submitted job. A reference by job name or pattern is only accepted if the referenced job is owned by the same user as the referring job. Each sub-task of the submitted job is not eligible for execution unless the corresponding sub-tasks of all jobs referenced in the comma-separated job id and/or job name list have completed.

The `wc_job_list` type is detailed in `sge_types(1)`.

Examples - Using Array Task Dependencies to Chunk Tasks

When using 3D rendering applications, it is often more efficient to render several frames at once on the same CPU instead of distributing the frames across several machines. The generation of several frames at once we will refer to as chunking.

Note: When using the task dependency facility, the array task must have the same range of sub-tasks as its dependent array task, otherwise the job will be rejected at submit time.

The following examples illustrate chunking:

- Array task B is dependent on array task A, which has a step size of 2:

  ```
  $ qsub -t 1-6:2 A
  $ qsub -hold_jid_ad A -t 1-6 B
  ```

  In the results shown below, it is assumed that array task A is chunking, which means that `B.1` and `B.2` are dependent on `A.1`, `B.3` and `B.4` are dependent on `A.3`, and so on. If job `A.1` didn't render frame 2, then job `B.2` would fail:

  ```
  | A.1 | --> | B.1 |
  |     | --> | B.2 |
  | A.3 | --> | B.3 |
  |     | --> | B.4 |
  | A.5 | --> | B.5 |
  |     | --> | B.6 |
  ```

- Array task B is dependent on array task A, which has a step size of 1:

  ```
  $ qsub -t 1-6 A
  $ qsub -hold_jid_ad A -t 1-6:2 B
  ```

  In this example shown below, array task B is chunking, which means that job `B.1` is dependent on job `A.1` and job `A.2`, job `B.3` is dependent on job `A.3` and job `A.4`, and so on. It is reasonable to always assume that array task B is chunking because otherwise `A.2`, `A.4`, and `A.6` would be needlessly run and the result would never be used:

  ```
  | A.1 | --> | B.1 |
  | A.2 | --> |     |
  | A.3 | --> | B.3 |
  | A.4 | --> |     |
  ```

```
| A.5 | --> | B.5 |
| A.6 | --> |     |
```

- Array task A has a step size of 3 and array task B has a step size of 2. The tasks are dependent on each other:

  ```
  $ qsub -t 1-6:3 A
  $ qsub -hold_jid_ad A -t 1-6:2 B
  ```

  In this example shown below, both array task A and array task B are chunking. So, job `B.1` is dependent on job `A.1`, job `B.3` is dependent on job `A.1` and job `A.4`, and job `B.5` is dependent on job `A.4`. When the hold array dependency option `-hold_jid_ad` is specified and the step sizes of the array job and the dependent array job are different, we always assume that both are chunking:

  ```
  | A.1 | --> | B.1 |
  |     | --> |     |
  |     | --> | B.3 |
  | A.4 | --> |     |
  |     | --> | B.5 |
  |     | --> |     |
  ```

Examples - Using Job Dependencies Versus Array Task Dependencies to Complete Array Jobs

The following example illustrates the difference between the job dependency facility and the task array dependency facility:

- In the following example, array task B is dependent on array task A:

  ```
  $ qsub -t 1-3 A
  $ qsub -hold_jid A -t 1-3 B
  ```

  All the sub-tasks in job B will wait for all sub-tasks 1, 2 and 3 in A to finish before starting the tasks in job B. The tasks will be executed in the following approximate order: `A.1`, `A.2`, `A.3`, `B.1`, `B.2`, `B.3`, as shown below:

  ```
  | A.1 |     | B.1 |
  | A.2 | --> | B.2 |
  | A.3 |     | B.3 |
  ```

- In the following example, each sub-task in array job B is dependent on each corresponding sub-task in job A in a one-to-one mapping:

  ```
  $ qsub -t 1-3 A
  $ qsub -hold_jid_ad A -t 1-3 B
  ```

  Sub-task `B.1` will only start when `A.1` completes. `B.2` will only start once `A.2` completes, etc. On a single machine renderfarm, the tasks thus could be executed in the following approximate order: `A.1`, `B.1`, `A.2`, `B.2`, `A.3`, `B.3`, as shown below:

  ```
  | A.1 | --> | B.1 |
  | A.2 | --> | B.2 |
  | A.3 | --> | B.3 |
  ```

  It should only be able to specify the option if we are submitting an array job, it is dependent on another array job, and that array job has the same number of sub-tasks.

2.7.2 How to Submit an Array Job From the Command Line

To submit an array job from the command line, type the following command:

```
qsub  -t <n[-m[:s]]> <job.sh>
```

The `-t` option defines the task index range. The `n[-m[:s]]` argument indicates the following:

- The lowest index number (n)
- The highest index number (m)
- The step size (s)

The range may be a single number (n), a simple range (n-m), or a range with a step size (n-m:s).

Example - Array Job

The following is an example of how to submit an array job:

```
% qsub -l h_cpu=0:45:0 -t 2-10:2 render.sh data.in
```

Each task requests a hard CPU time limit of 45 minutes with the `-l` option. The `-t` option defines the task index range. In this case, `2-10:2` specifies that 2 is the lowest index number, and 10 is the highest index number. Only every second index, the :2 part of the specification, is used. Thus, the array job is made up of 5 tasks with the task indices 2, 4, 6, 8, and 10. Each task executes the job script render.sh once the task is dispatched and started by the Grid Engine system. Tasks can use `SGE_TASK_ID` to find their index number, which they can use to find their input data record in the data file `data.in`.

2.7.3 How to Submit an Array Job With QMON

To submit an array job, follow the instructions in How to Submit a Simple Job With QMON, additionally taking into account the following information.

The only difference is that the Job Tasks input window that is shown in Figure 2–8 must contain the task range specification. The task range specification uses syntax that is identical to the `qsub -t` command.

For information about monitoring and controlling jobs in general, and about array jobs in particular, see Monitoring and Controlling Jobs.

Note: Array tasks cannot have interdependencies with other jobs or with other array tasks.

2.8 Submitting Interactive Jobs

The submission of interactive jobs instead of batch jobs is useful in situations where a job requires your direct input to influence the job results. Such situations are typical for X Windows applications or for tasks in which your interpretation of immediate results is required to steer further processing.

You can create interactive jobs in three ways:

- `qlogin` - An rlogin-like session that is started on a host selected by the Grid Engine software.

- `qrsh` - The equivalent of the standard UNIX `rsh` facility. A command is run remotely on a host selected by the Grid Engine system. If no command is specified, a remote rlogin session is started on a remote host.
- `qsh` - An `xterm` that is displayed from the machine that is running the job. The display is set corresponding to your specification or to the setting of the DISPLAY environment variable. If the DISPLAY variable is not set, and if no display destination is defined, the Grid Engine system directs the `xterm` to the 0.0 screen of the X server on the host from which the job was submitted.

Note: Contact your system administrator to find out if your cluster is prepared for interactive job execution. To function correctly, all the facilities need proper configuration of cluster parameters of the Grid Engine system. The correct `xterm` execution paths must be defined for `qsh`. Interactive queues must be available for this type of job.

The default handling of interactive jobs differs from the handling of batch jobs. Interactive jobs are not queued if the jobs cannot be executed when they are submitted. When a job is not queued immediately, the user is notified that the cluster is currently too busy.

You can change this default behavior with the `-now no` option to `qsh`, `qlogin`, and `qrsh`. If you use this option, interactive jobs are queued like batch jobs. When you use the `-now yes` option, batch jobs that are submitted with `qsub` can also be handled like interactive jobs. Such batch jobs are either dispatched for running immediately, or they are rejected.

Note: Interactive jobs can be run only in queues of the type INTERACTIVE. See *Oracle Grid Engine Administration Guide* for configuring queues details.

The following sections describe how to use the `qlogin` and `qsh` facilities. The `qrsh` command is explained in a broader context in Transparent Remote Execution.

2.8.1 How to Submit Interactive Jobs From the Command Line

Note: The output for an interactive job cannot be redirected with the `-j y|n, -o`, and `-e` options. However, since the output for a prolog and epilog script is sent to the default `stdout` and `stderr` files, you can use the `-j y|n, -o`, and `-e` options to redirect this output to different files.

2.8.1.1 Using qrsh to Submit Interactive Jobs

`qrsh` supports most of the `qsub` options. If no options are given, `qrsh` will open an rlogin-like session.

To submit an interactive job with the `qrsh` command, type a command like the following:

```
qrsh -pty y vi
```

This command starts a vi editor on any available system in the Grid Engine cluster. The `-pty y` option starts a job in a pseudo-terminal session. The pseudo-terminal allows full cursor control from within the vi session.

The CTRL-Z behavior of `qrsh <jobname>` is since version 6.2u7 controllable with the parameter:

```
qrsh -suspend_remote y[es]| n[o] <your_program>
```

When your desired behavior is that you want to suspend `qrsh` and the submitted job, when you press CTRL-Z, then you have to submit the job like the following:

```
qrsh -suspend_remote yes <your_program>
```

If you just want to suspend `qrsh` and let the remote program in running state, then you have to use:

```
qrsh -suspend_remote no <your_program>
```

Note: If you have submitted a job in one way but during run time you want to have the opposite behavior than specified, you can press the % key and afterwords the CTRL-Z key.

2.8.1.2 Using qsh to Submit Interactive Jobs

`qsh` is very similar to `qsub`. `qsh` supports several of the `qsub` options, as well as the additional option `-display` to direct the display of the `xterm` to be invoked.

To submit an interactive job with the `qsh` command, type a command like the following:

```
% qsh -l arch=solaris64
```

This command starts an `xterm` on any available Sun Solaris 64-bit operating system host.

2.8.1.3 Using qlogin to Submit Interactive Jobs

Use the `qlogin` command from any terminal window to start an interactive session under the control of the Grid Engine system.

To submit an interactive job with the `qlogin` command, type a command like the following:

```
% qlogin -l star-cd=1,h_cpu=6:0:0
```

This command locates a low-loaded host. The host has a Star-CD license available. The host also has at least one queue that can provide a minimum of six hours hard CPU time limit.

Note: Depending on the remote login facility that is configured to be used by the Grid Engine system, you might have to provide your user name, your password, or both, at a login prompt.

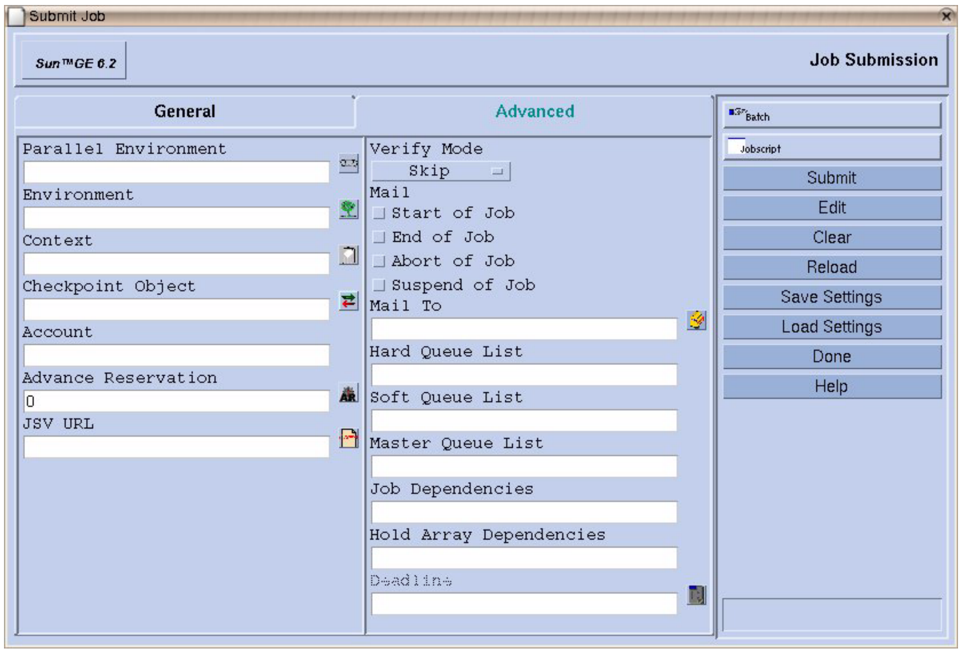
2.8.2 How to Submit Interactive Jobs With QMON

Note: The only type of interactive jobs that you can submit from QMON are jobs that bring up an `xterm` on a host selected by the Grid Engine system.

---

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select the Submit Jobs button.
3. Verify that the top button on the right side of the dialog box says "Interactive". If not, click the button to change from Batch to Interactive. This prepares the Submit Job dialog box to submit interactive jobs. The meaning and the use of the selection options in the dialog box is almost the same as that described for batch jobs in Submitting Batch Jobs. The difference is that several input fields are grayed out because those fields do not apply to interactive jobs. The following figures show the general and advanced variations of the Interactive Submit Job dialog box.

Figure 2–2 General Options for Job Submission



Submit Job

Sun™GE 6.2

Job Submission

General | Advanced

Parallel Environment

Environment

Context

Checkpoint Object

Account

Advance Reservation
0

JSV URL

Verify Mode
Skip

Mail
Start of Job
End of Job
Abort of Job
Suspend of Job

Mail To

Hard Queue List

Soft Queue List

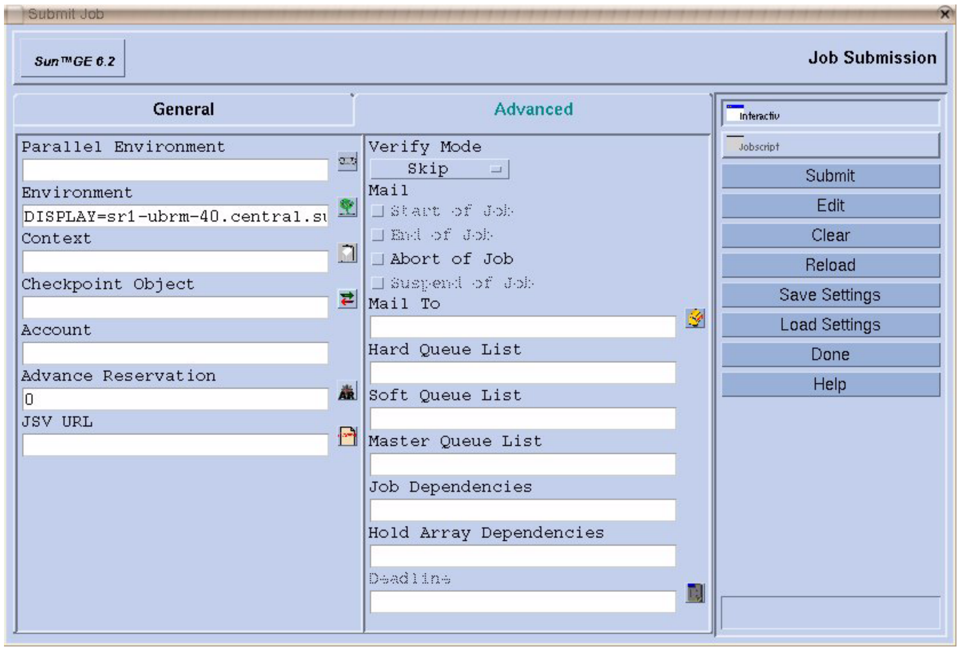
Master Queue List

Job Dependencies

Hold Array Dependencies

Deadline

Figure 2–3 Advanced Options for Job Submission



Parallel Environment

Environment
DISPLAY=sr1-ubrm-40.central.su

Context

Checkpoint Object

Account

Advance Reservation
0

JSV URL

Verify Mode
Skip

Mail
Start of Job
End of Job
Abort of Job
Suspend of Job

Mail To

Hard Queue List

Soft Queue List

Master Queue List

Job Dependencies

Hold Array Dependencies

Deadline

2.9 Transparent Remote Execution

The Grid Engine system provides a set of closely related facilities that support the transparent remote execution of certain computational tasks. The core tool for this functionality is the `qrsh` command, which is described in Remote Execution With qrsh. Two high-level facilities, `qtcsh` and `qmake`, build on top of `qrsh`. These two commands enable the Grid Engine system to transparently distribute implicit computational tasks, thereby enhancing the standard UNIX facilities `make` and `csh`.

2.9.1 Remote Execution With qrsh

`qrsh` is the major enabling infrastructure for the implementation of the `qtcsh` and the `qmake` facilities. `qrsh` is also used for the tight integration of the Grid Engine system with parallel environments such as MPI or PVM.

You can use `qrsh` for various purposes, including the following:

- To provide remote execution of interactive applications that use the Grid Engine system. This is comparable to the standard UNIX facility `rsh`, which is also called `remsh` on HP-UX systems.
- To offer interactive login session capabilities that use the Grid Engine system. By default, `qlogin` is similar to the standard UNIX facility rlogin but it can also be configured to use the UNIX telnet facility or any similar remote login facility.
- To allow for the submission of batch jobs that support terminal I/O (standard output, standard error, and standard input) and terminal control.
- To provide a way to submit a standalone program that is not embedded in a shell script.

Note: You can also submit scripts with `qrsh` by using the `-b n` option.

- To provide a submission client that remains active while a batch job is pending or running and that goes away only if the job finishes or is cancelled.
- To allow for the Grid Engine system-controlled remote running of job tasks within the framework of the dispersed resources allocated by parallel jobs. See *Oracle Grid Engine Administration Guide* for more information about tight integration of parallel environments and Oracle Grid Engine software.

2.9.1.1 Invoking Transparent Remote Execution With qrsh

Type the `qrsh` command, adding options and arguments according to the following syntax:

```
% qrsh    [<options>] <program>|<shell-script> [<arguments>] \
    [> stdout] [>&2 stderr] [< stdin]
```

`qrsh` understands almost all options of `qsub`. `qrsh` provides the following options:

- `-now yes|no` - now yes specifies that the job is scheduled immediately. The job is rejected if no appropriate resources are available. `-now yes` is the default. `-now no` specifies that the job is queued like a batch job if the job cannot be started at submission time.
- `-inherit` - `qrsh` does not go through the scheduling process to start a job-task. Instead, `qrsh` assumes that the job is embedded in a parallel job that already has allocated suitable resources on the designated remote execution host. This form of `qrsh` is commonly used in `qmake` and in a tight parallel environment integration. The default is not to inherit external job resources.
- `-binary yes|no` - When specified with the n option, enables you to use `qrsh` to submit script jobs.
- `-noshell` - With this option, you do not start the command line that is given to `qrsh` in a user's login shell. Instead, you execute the command without the wrapping shell. Use this option to speed up execution.
- `-nostdin` - Suppresses the input stream STDIN. With this option set, `qrsh` passes the `-n` option to the `rsh` command. Suppression of the input stream is especially useful if multiple tasks are executed in parallel using `qrsh`, for example, in a make process. It is undefined which process gets the input.
- `-pty yes|no` - Available for `qrsh`, `qlogin`, and `qsub` only, `-pty yes` starts the job in a pseudo terminal (`pty`). If no `pty` is available, the job fails to start. `-pty no` starts the job without a pseudo terminal. By default, `qrsh` without a command and `qlogin` start the job in a `pty`, `qrsh` with a command starts the job without a `pty`. If a job is running in a `pty`, you can suspend the client by entering CTRL-Z. CTRL-Z will only suspend the remote job started within the IJS session.
- `-verbose` - This option presents output on the scheduling process. Verbose is mainly intended for debugging purposes and is switched off by default.

2.9.2 Transparent Job Distribution With qtcsh

`qtcsh` is a fully compatible replacement for the widely known and used UNIX C shell derivative `tcsh`. `qtcsh` is built around `tcsh`. See the information that is provided in `$SGE_ROOT/3rd_party` for details on the involvement of `tcsh`.

`qtcsh` provides a command shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts that use the Grid Engine system. The `.qtask` configuration files define the applications to execute remotely and the requirements that apply to the selection of an execution host.

These applications are transparent to the user and are submitted to the Grid Engine system through the `qrsh` facility. `qrsh` provides standard output, error output, and standard input handling as well as terminal control connection to the remotely executing application.

Three noticeable differences between running such an application remotely and running the application on the same host as the shell are:

- The remote host might be more powerful, lower-loaded, and have required hardware and software resources installed.
- A small delay is incurred by the remote startup of the jobs and by their handling through the Grid Engine system.
- Administrators can restrict the use of resources through interactive jobs (`qrsh`) and thus through `qtcsh`. If not enough suitable resources are available for an application to be started through `qrsh`, or if all suitable systems are overloaded, the implicit `qrsh` submission fails. A corresponding error message is returned, such as `Not enough resources ... try later.`

In addition to the standard use, `qtcsh` is a suitable platform for third-party code and tool integration. The single-application execution form of `qtcsh` is `qtcsh -c app-name`. The use of this form of `qtcsh` inside integration environments presents a persistent interface that almost never needs to be changed. All the required application, tool, integration, site, and even user-specific configurations are contained in appropriately defined `.qtask` files. A further advantage is that this interface can be used in shell scripts, in C programs, and even in Java applications.

2.9.2.1 qtcsh Usage

The invocation of `qtcsh` is exactly the same as for `tcsh`. `qtcsh` extends tcsh by providing support for the `.qtask` file and by offering a set of specialized shell built-in modes.

The `.qtask` file is defined as follows. Each line in the file has the following format:

```
% [!]<app-name> <qrsh-options>
```

The optional leading exclamation mark (!) defines the precedence between conflicting definitions in a global cluster `.qtask` file and the personal `.qtask` file of the `qtcsh` user. If the exclamation mark is missing in the global cluster file, a conflicting definition in the user file overrides the definition in the global cluster file. If the exclamation mark is in the global cluster file, the corresponding definition cannot be overridden.

`app-name` specifies the name of the application that is submitted to the Grid Engine system for remote execution. The application name must appear in the command line exactly as the application is defined in the `.qtask` file. If the application name is prefixed with a path name, a local binary is addressed. No remote execution is intended.

`qrsh-options` specifies the options to the `qrsh` facility to use. These options define resource requirements for the application.

`csh` aliases are expanded before a comparison with the application names is performed. The applications intended for remote execution can also appear anywhere in a `qtcsh` command line, in particular before or after standard I/O redirections.

The following examples demonstrate the syntax:

```
# .qtask file
netscape -v DISPLAY=myhost:0
```

```
grep -l h=filesurfer
```

Given this `.qtask` file, the following `qtcsh` command lines:

```
netscape
~/mybin/netscape
cat very_big_file | grep pattern | sort | uniq
```

Result in:

```
qrsh -v DISPLAY=myhost:0 netscape
~/mybin/netscape
cat very_big_file | qrsh -l h=filesurfer grep pattern | sort | uniq
```

`qtcsh` can operate in different modes, influenced by switches that can be set on or off:

- Local or remote execution of commands. Remote is the default.
- Immediate or batch remote execution. Immediate is the default.
- Verbose or nonverbose output. Nonverbose is the default.

The setting of these modes can be changed using option arguments of `qtcsh` at start time or with the shell built-in command `qrshmode` at runtime.

2.9.3 Parallel Makefile Processing With qmake

`qmake` is a replacement for the standard UNIX make facility. `qmake` extends make by enabling the distribution of independent make steps across a cluster of suitable machines. `qmake` is built around the popular GNU-make facility `gmake`. See the information that is provided in `$SGE_ROOT/3rd_party` for details on the involvement of `gmake`.

To ensure that a distributed make process can run to completion, `qmake` does the following:

1. Allocates the required resources in a way analogous to a parallel job.
2. Manages this set of resources without further interaction with the scheduling.
3. Distributes make steps as resources become available, using the `qrsh` facility with the `-inherit` option.

`qrsh` provides standard output, error output, and standard input handling as well as terminal control connection to the remotely executing make step. There are only three noticeable differences exist between executing a make procedure locally and using `qmake`:

- The parallelization of the make process will speed up significantly, provided that individual make steps have a certain duration and that enough independent make steps exist to process.
- In the make steps to be started up remotely, an implied small overhead exists that is caused by `qrsh` and the remote execution.
- To take advantage of the make step distribution of `qmake`, the user must specify as a minimum the degree of parallelization. That is, the user must specify the number of concurrently executable make steps. In addition, the user can specify the resource characteristics required by the make steps, such as available software licenses, machine architecture, memory, or CPU-time requirements.

The most common use of make is the compilation of complex software packages. However, compilation might not be the major application for `qmake`. Program files are often quite small as a matter of good programming practice. Therefore, compilation of a single program file, which is a single make step, often takes only a few seconds. Furthermore, compilation usually implies significant file access. Nested include files can cause this problem. File access might not be accelerated if done for multiple make steps in parallel because the file server can become a bottleneck. Such a bottleneck effectively serializes all the file access. Therefore, the compilation process sometimes cannot be accelerated in a satisfactory manner.

Other potential applications of `qmake` are more appropriate. An example is the steering of the interdependencies and the workflow of complex analysis tasks through makefiles. Each make step in such environments is typically a simulation or data analysis operation with nonnegligible resource and computation time requirements. A considerable acceleration can be achieved in such cases.

2.9.3.1 qmake Usage

The command-line syntax of `qmake` looks similar to the syntax of `qrsh`:

```
% qmake [-pe <pe-name pe-range>][<options>] \
 -- [<gnu-make-options>][<target>]
```

Note: The `-inherit` option is also supported by `qmake`, as described later in this section.

Pay special attention to the use of the `-pe` option and its relation to the `gmake -j` option. You can use both options to express the amount of parallelism to be achieved. The difference is that `gmake` provides no possibility with `-j` to specify something like a parallel environment to use. Therefore, `qmake` assumes that a default environment for parallel makes is configured that is called make. Furthermore, gmake's `-j` allows for no specification of a range, but only for a single number. `qmake` interprets the number that is given with `-j` as a range of 1n. By contrast, `-pe` permits the detailed specification of all these parameters. Consequently the following command line examples are identical:

```
% qmake -- -j 10
% qmake -pe make 1-10 --
```

The following command lines cannot be expressed using the `-j` option:

```
% qmake -pe make 5-10,16 --
% qmake -pe mpi 1-99999 --
```

Apart from the syntax, `qmake` supports two modes of invocation: interactively from the command line without the `-inherit` option, or within a batch job with the `-inherit` option. These two modes start different sequences of actions:

- Interactive - When `qmake` is invoked on the command line, the make process is implicitly submitted to the Grid Engine system with `qrsh`. The process is as follows:
    1. The resource requirements that are specified in the `qmake` command line are taken into account.
    2. The Grid Engine system selects a master machine for the execution of the parallel job that is associated with the parallel make job.
    3. The Grid Engine system starts the make procedure. The procedure must start there because the make process can be architecture-dependent. The required architecture is specified in the `qmake` command line.

4. The `qmake` process on the master machine delegates execution of individual make steps to the other hosts that are allocated for the job. The steps are passed to `qmake` through the parallel environment hosts file.

- Batch - In this case, `qmake` appears inside a batch script with the `-inherit` option. If the `-inherit` option is not present, a new job is spawned, as described in the first case earlier. This results in `qmake` making use of the resources already allocated to the job into which `qmake` is embedded. `qmake` uses `qrsh -inherit` directly to start make steps. When calling `qmake` in batch mode, the specification of resource requirements, the `-pe` option and the `-j` option are ignored.

Note: Single CPU jobs also must request a parallel environment:

```
qmake -pe make 1 --
```

If no parallel execution is required, call `qmake` with `gmake` command-line syntax without Grid Engine system options and without --. This `qmake` command behaves like `gmake`.

2.10 How to Submit a Simple Job From the Command Line

Before You Begin

Note: If you installed the Grid Engine software under an unprivileged user account, you must log in as that user to be able to run jobs. See *Oracle Grid Engine Installation and Upgrade Guide* for information about installation accounts.

Before you run any Grid Engine system command, you must first set your executable search path and other environment conditions properly.

Steps

1. From the command line, type one of the following commands:

   - If you are using `csh` or `tcsh` as your command interpreter, type the following:

     ```
     % source $SGE_ROOT/$SGE_CELL/common/settings.csh
     ```

     `$SGE_ROOT` specifies the location of the root directory of the Grid Engine system. This directory was specified at the beginning of the installation procedure.

   - If you are using `sh`, `ksh`, or `bash` as your command interpreter, type the following:

     ```
     # . $SGE_ROOT/$SGE_CELL/common/settings.sh
     ```

Note: You can add these commands to your `.login`, `.cshrc`, or `.profile` files, whichever is appropriate. By adding these commands, you guarantee proper settings for all interactive session you start later.

2. Submit a simple job script to your cluster by typing the following command:

   ```
   % qsub simple.sh
   ```

   The command assumes that `simple.sh` is the name of the script file, and that the file is located in your current working directory. You can find the following job in the file `$SGE_ROOT/examples/jobs/simple.sh`.

   ```
   #!/bin/sh
   #
   #
   # (c) 2004 Sun Microsystems, Inc. Use is subject to license terms.

   # This is a simple example of a Grid Engine batch script

   # request Bourne shell as shell for job
   #$ -S /bin/sh

   #
   # print date and time
   date
   # Sleep for 20 seconds
   sleep 20
   # print date and time again
   date
   ```

   If the job submits successfully, the `qsub` command responds with a message similar to the following example:

   ```
   your job 1 ('simple.sh') has been submitted
   ```

3. Type the following command to retrieve status information about your job.

   ```
   % qstat
   ```

   You should receive a status report that provides information about all jobs currently known to the Grid Engine system. For each job, the status report lists the following items:

   - Job ID, which is the unique number that is included in the submit confirmation
   - Name of the job script
   - Owner of the job
   - State indicator; for example, r means running
   - Submit or start time
   - Name of the queue in which the job runs

If `qstat` produces no output, no jobs are actually known to the system. For example, your job might already have finished.

You can control the output of the finished jobs by checking their `stdout` and `stderr` redirection files. By default, these files are generated in the job owner's home directory on the host that ran the job. The names of the files are composed of the job script file name with a `.o` extension for the `stdout` file and a `.e` extension for the `stderr` file, followed by the unique job ID. The `stdout` and `stderr` files of your job can be found under the names `simple.sh.o1` and `simple.sh.e1` respectively. These names are used if your job was the first ever executed in a newly installed Grid Engine system.

2.11 How to Submit a Simple Job With QMON

Before You Begin

Note: If you installed the Grid Engine software under an unprivileged user account, you must log in as that user to be able to run jobs. See *Oracle Grid Engine Installation and Upgrade Guide* for details about installation accounts.

A more convenient way to submit and control jobs and of getting an overview of the Grid Engine system is the graphical user interface QMON. Among other facilities, QMON provides a job submission dialog box and a Job Control dialog box for the tasks of submitting and monitoring jobs.

Steps

1. Type the following command to launch QMON:

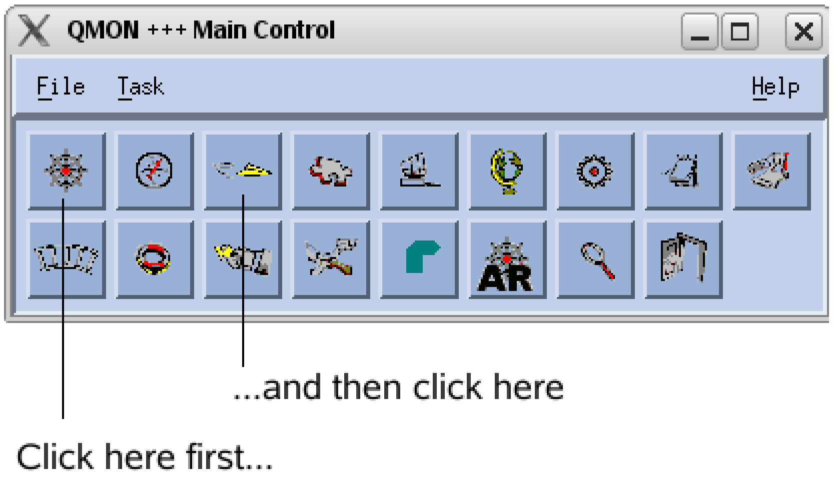
   ```
   % qmon
   ```

   During startup, a message window appears, and then the QMON Main Control window appears.

2. Click the Job Control button, and then click the Submit Jobs button, as shown below.

   Tip: The button names, such as Job Control, are displayed when you rest the mouse pointer over the buttons.

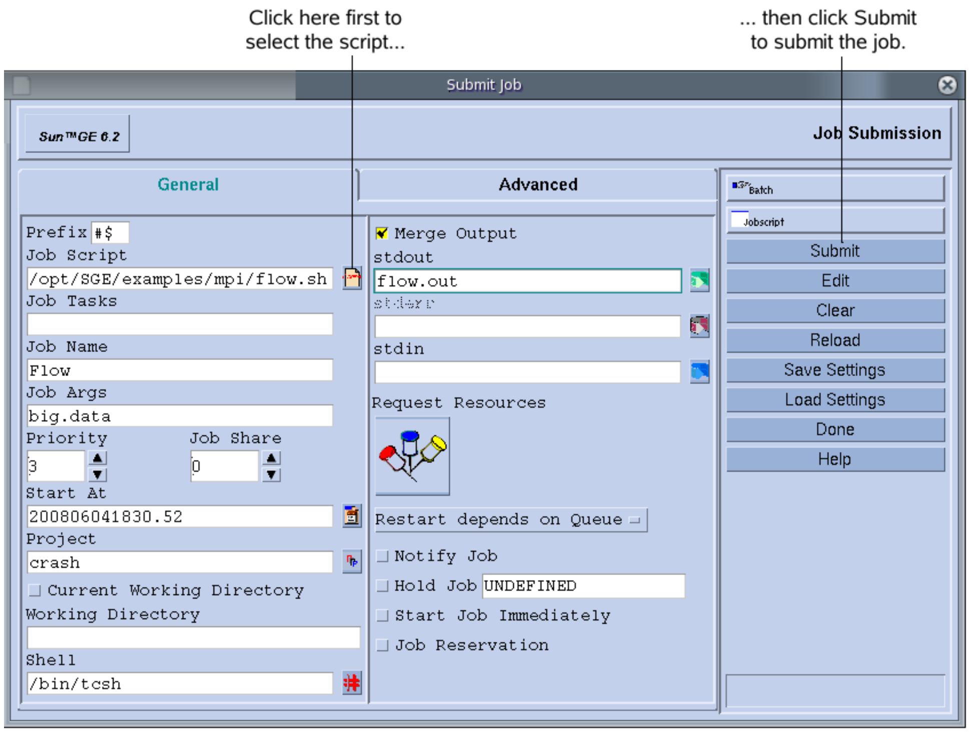
Figure 2–4 QMON Main Control



   The Job Control and Submit Job dialog boxes.

3. In the Submit Job dialog box, click the icon at the right of the Job Script field. The Select a File dialog box appears.

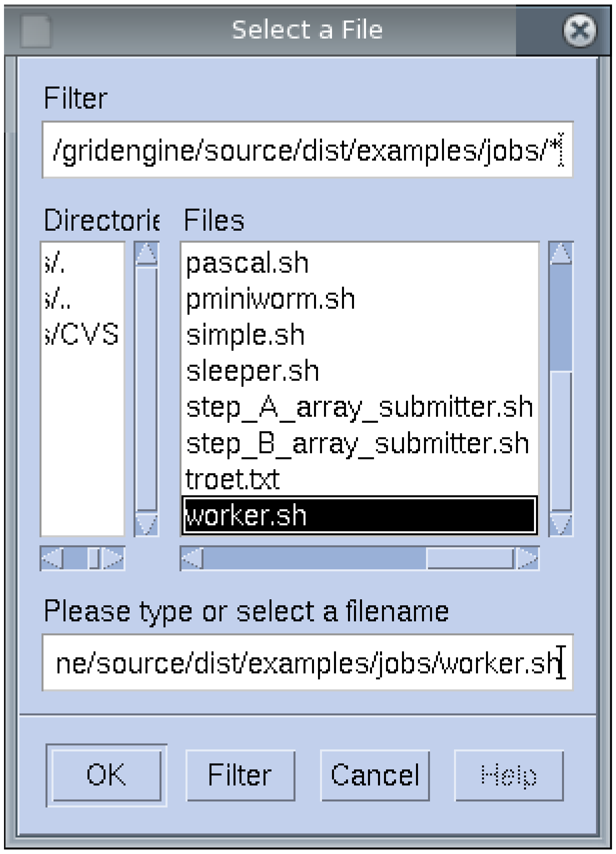
Figure 2–5 Job Submission Window



Prefix #$
Job Script
/opt/SGE/examples/mpi/flow.sh
Job Tasks
Job Name
Flow
Job Args
big.data
Priority 3
Job Share 0
Start At
200806041830.52
Project
crash
Current Working Directory
Working Directory
Shell
/bin/tcsh

Merge Output ✓
stdout
flow.out
stderr
stdin
Request Resources
Restart depends on Queue
Notify Job
Hold Job UNDEFINED
Start Job Immediately
Job Reservation

Figure 2–6 Select Script Window



Filter
/gridengine/source/dist/examples/jobs/*

Directorie
s/.
s/..
s/CVS

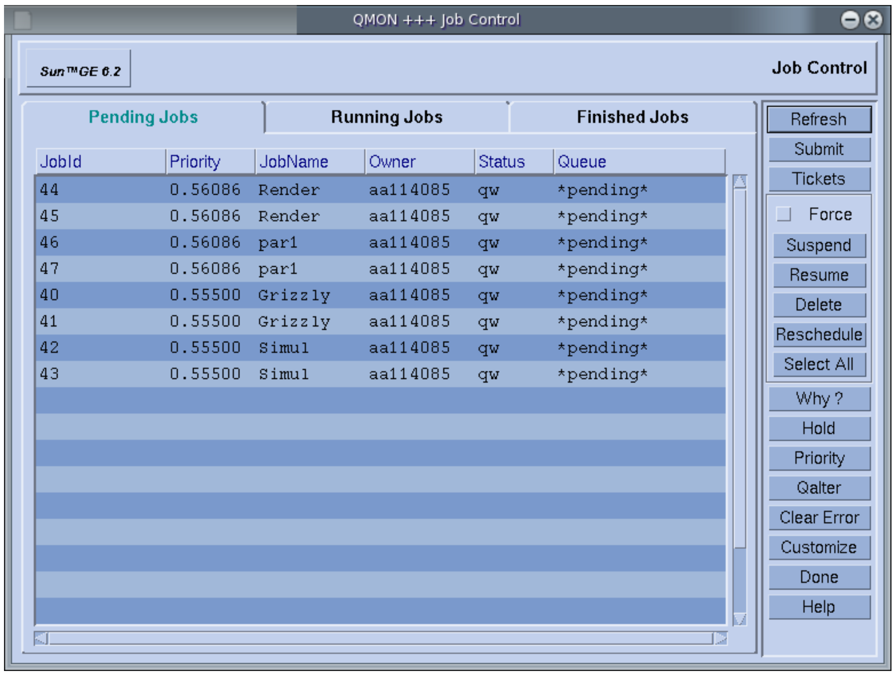
Files
pascal.sh
pminiworm.sh
simple.sh
sleeper.sh
step_A_array_submitter.sh
step_B_array_submitter.sh
troet.txt
worker.sh

Please type or select a filename
ne/source/dist/examples/jobs/worker.sh

4. Select your script file. For example, select the file simple.sh that was used in the command line example.
5. Click OK to close the Select a File dialog box.
6. On the Submit Job dialog box, click Submit.

   After a few seconds, you should be able to monitor your job on the Job Control dialog box. First you see your job on the Pending Jobs tab. Once the job starts running, the job quickly moves to the Running Jobs tab.

Figure 2–7 QMON Job Control Window



JobId	Priority	JobName	Owner	Status	Queue
44	0.56086	Render	aa114085	qw	*pending*
45	0.56086	Render	aa114085	qw	*pending*
46	0.56086	par1	aa114085	qw	*pending*
47	0.56086	par1	aa114085	qw	*pending*
40	0.55500	Grizzly	aa114085	qw	*pending*
41	0.55500	Grizzly	aa114085	qw	*pending*
42	0.55500	Simul	aa114085	qw	*pending*
43	0.55500	Simul	aa114085	qw	*pending*

2.12 How to Submit an Extended Job From the Command Line

To submit the extended job request that is shown in Figure 2–8 from the command line, type the following command:

```
% qsub -N Flow -p -111 -P devel -a 200404221630.44 -cwd \
    -S /bin/tcsh -o flow.out -j y flow.sh big.data
```

2.13 How to Submit an Extended Job With QMON

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select a pending job and click the Submit button. The Submit Job dialog box appears. See the example below. The General tab of the Submit Job dialog box enables you to configure the following parameters for an extended job:
   - Prefix - A prefix string that is used for script-embedded submit options. See Active Comments for details.
   - Job Script - The job script to use. Click the icon at the right of the Job Script field to open a file selection box.

- Job Tasks - The task ID range for submitting array jobs. See Submitting Array Jobs for details.
- Job Name - The name of the job. A default is set after you select a job script.
- Job Args - Arguments to the job script.
- Priority - A counting box for setting the job's initial priority This priority ranks a single user's jobs. Priority tells the scheduler how to choose among a single user's jobs when several of that user's jobs are in the system simultaneously.

  Note: To enable users to set the priorities of their own jobs, the administrator must enable priorities with the 4 parameter of the scheduler configuration. For more information about managing policies, see *Oracle Grid Engine Administrator Guide*.

- Job Share - Defines the share of the job's tickets relative to other jobs. The job share influences only the share tree policy and the functional policy.
- Start At - The time at which the job is considered eligible for execution. Click the icon at the right of the Start At field to open a dialog box.
- Project - The project to which the job is subordinated. Click the icon at the right of the Project field to select among the available projects.
- Current Working Directory - A flag that indicates whether to execute the job in the current working directory. Use this flag only for identical directory hierarchies between the submit host and the potential execution hosts.
- Shell - The command interpreter to use to run the job script. See How a Command Interpreter is Selected for details. Click the icon at the right of the Shell field to open a dialog box. Enter the command interpreter specifications of the job.
- Merge Output - A flag indicating whether to merge the job's standard output and standard error output together into the standard output stream.
- stdout - The standard output redirection to use. See Output Redirection for details. A default is used if nothing is specified. Click the icon at the right of the stdout field to open a dialog box. Enter the output redirection alternatives.
- stderr - The standard error output redirection to use, similar to the standard output redirection.
- stdin - The standard input file to use, similar to the standard output redirection.
- Request Resources - Click this button to define the resource requirement for your job. If resources are requested for a job, the button changes color.
- Restart depends on Queue - Click this button to define whether the job can be restarted after being aborted by a system crash or similar events. This button also controls whether the restart behavior depends on the queue or is demanded by the job.
- Notify Job - A flag that indicates whether the job is to be notified by SIGUSR1 or by SIGUSR2 signals if the job is about to be suspended or canceled.
- Hold Job - A flag that indicates either a user hold or a job dependency is to be assigned to the job. The job is not eligible for execution as long as any type of hold is assigned to the job. See Monitoring and Controlling Jobs for more

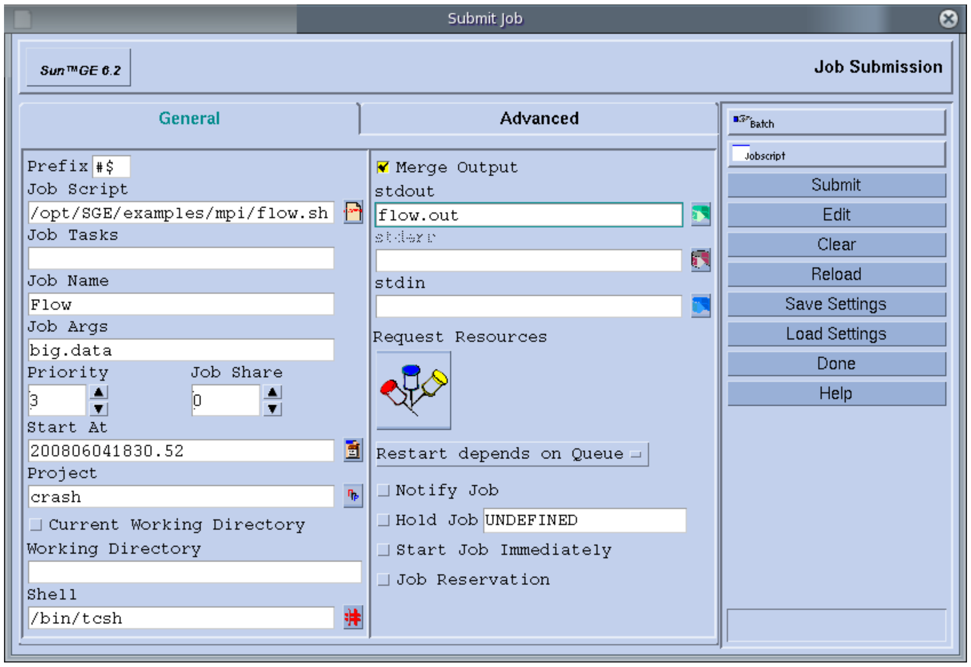
details. To restrict a hold, enter a specific range of tasks for an array job in the Hold Job field. For more information, see Submitting Array Jobs.

- Start Job Immediately - A flag that forces the job to be started immediately, if possible, or to be rejected. Jobs are not queued if this flag is selected.
- Job Reservation - A flag that specifies which resources should be reserved for a job. See *Oracle Grid Engine Administration Guide* for resource reservation and backfiling. The buttons at the right side of the Submit Job dialog box enable you to start various actions:
- Submit - Submit the currently specified job.
- Edit - Edit the selected script file in an X terminal, using either vi or the editor defined by the EDITOR environment variable.
- Clear - Clear all settings in the Submit Job dialog box, including any specified resource requests.
- Reload - Reload the specified script file, parse any script-embedded options, parse default settings, and discard intermediate manual changes to these settings. For more information, see Active Comments and Default Request Files. This action is the equivalent to a Clear action with subsequent specifications of the previous script file. The option has an effect only if a script file is already selected.
- Save Settings - Save the current settings to a file. Use the file selection box to select the file. The saved files can either be loaded later or be used as default requests. For more information, Default Request Files.
- Load Settings - Load settings previously saved with the Save Settings button. The loaded settings overwrite the current settings.
- Done - Closes the Submit Job dialog box.

Example - Extended Job Example

The following figure shows the Submit Job dialog box with most of the parameters set.

Figure 2–8 Extended Job Example


Submit Job

Sun™GE 6.2 Job Submission

General Advanced

Prefix #$
Job Script /opt/SGE/examples/mpi/flow.sh
Job Tasks
Job Name Flow
Job Args big.data
Priority 3
Job Share 0
Start At 200806041830.52
Project crash
Current Working Directory
Working Directory
Shell /bin/tcsh

Merge Output ✓
stdout flow.out
stderr
stdin
Request Resources
Restart depends on Queue
Notify Job
Hold Job UNDEFINED
Start Job Immediately
Job Reservation

The parameters of the job configured in the example are:

- The job has the script file `flow.sh`, which must reside in the working directory of QMON.
- The job is called `Flow`.
- The script file takes the single argument `big.data`.
- The job starts with priority 3.
- The job is eligible for execution not before 4:30.44 AM of the 22th of April in the year 2004.
- The project definition means that the job is subordinated to project `crash`.
- The job is executed in the submission working directory.
- The job uses the `tcsh` command interpreter.
- Standard output and standard error output are merged into the file `flow.out`, which is created in the current working directory.

2.14 How to Submit an Advanced Job From the Command Line

To submit the advanced job request that is shown in Figure 2–9 from the command line, type the following command:

```
% qsub -N Flow -p -111 -P devel -a 200012240000.00 -cwd \
 -S /bin/tcsh -o flow.out -j y -pe mpi 4-16 \
 -v SHARED_MEM=TRUE,MODEL_SIZE=LARGE \
 -ac JOB_STEP=preprocessing,PORT=1234 \
 -A FLOW -w w -m s,e -q big_q\
 -M me@myhost.com,me@other.address \
 flow.sh big.data
```

2.14.1 Specifying the Use of a Script or a Binary

Note: Submitting a command as a script can add a number of operations to the submission process and have a negative impact on performance. This impact can be significant if you have short running jobs and big job scripts. If job scripts are available on the execution nodes, that is through NFS, binary submission may be a better choice.

You can use the `-b n|y` submit option to indicate explicitly whether the command should be treated as a binary or a script:

- To specify that the command should be treated as a binary or a script, use the `-b y` option with the `qrsh` command.
- To specify the command should be treated only as a script, use the `-b n` option with the `qsub` command.

2.14.2 Default Request Files

The preceding command shows that advanced job requests can be complex, especially if similar requests need to be submitted frequently. To avoid these problems, you can embed `qsub` options in the script files, or use default request files. For more information, see Active Comments.

The cluster administrator can set up a global default request file for all Grid Engine system users. Users can define a private default request file located in their home directories. In addition, users can create application specific default request files.

If more than one of these files are available, the files are merged into one default request, with the following order of precedence:

1. Application-specific default request file
2. General private default request file
3. Global default request file

Default request files contain the `qsub` options to apply by default to the jobs in one or more lines. The location of the global cluster default request file is `$SGE_ROOT/cell/common/sge_request`. The private general default request file is located under `$HOME/.sge_request`. The application-specific default request files are located under `$cwd/.sge_request`.

Script embedding and the `qsub` command line have higher precedence than the default request files. Therefore, script embedding overrides default request file settings. The `qsub` command line options can override these settings again.

To discard any previous settings, use the `qsub -clear` command in a default request file, in embedded script commands, or in the `qsub` command line.

Example - Private Default Request File

Here is an example of a private default request file:

```
-A myproject -cwd -M me@myhost.com -m b e
-r y -j y -S /bin/ksh
```

Unless overridden, the following is true for all of this user's jobs:

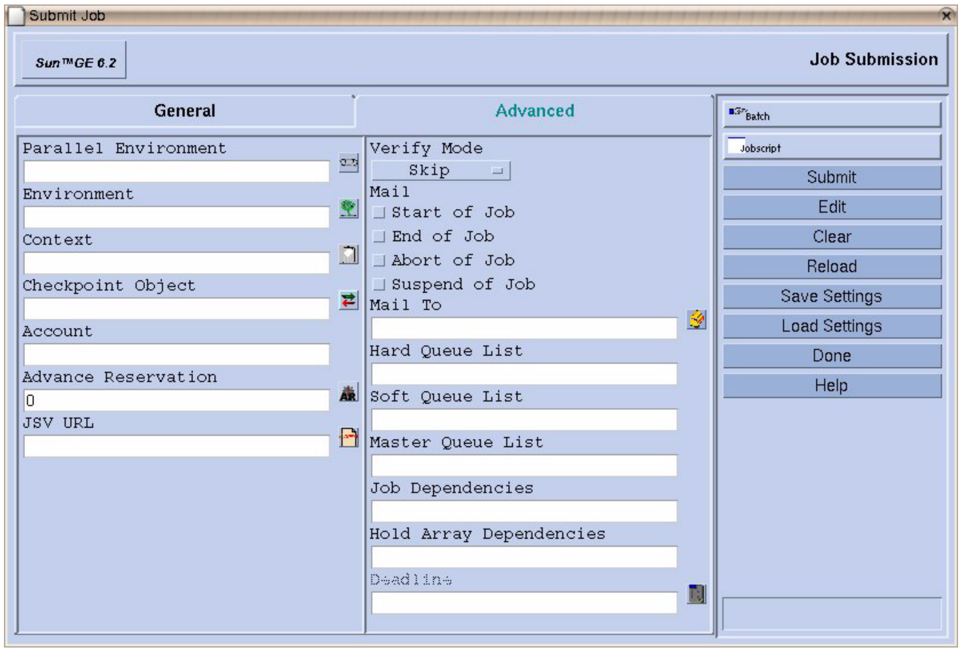
- The account string is myproject

- The jobs execute in the current working directory
- Mail notification is sent to me@myhost.com at the beginning and at the end of the jobs
- The standard output and standard error output are merged
- The `ksh` is used as command interpreter

2.15 How to Submit an Advanced Job With QMON

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select a pending job and click the Qalter button. The Submit Job dialog box appears.
3. Click the Advanced Tab, which shown below.

Figure 2–9 Advanced Job Example



The Advanced tab of the Submit Job dialog box enables you to define the following additional parameters:

- Parallel Environment - A list of available, configured parallel environments.
- Environment - A set of environment variables to set for the job before the job runs. Environment variables can be taken from QMON`s runtime environment, or you can define your own environment variables.
- Context - A list of name/value pairs that can be used to store and communicate job-related information. This information is accessible anywhere from within a cluster. You can modify context variables from the command line with the `-ac`, `-dc`, and `-sc` options to `qsub`, `qrsh`, `qsh`, `qlogin`, and `qalter`.

- Checkpoint Object - The checkpointing environment to use if checkpointing the job is desirable and suitable. See Monitoring and Controlling Jobs for details.
- Account - An account string to associate with the job. The account string is added to the accounting record that is kept for the job. The accounting record can be used for later accounting analysis.
- Verify Mode - The Verify flag determines the consistency checking mode for your job. To check for consistency of the job request, the Grid Engine system assumes an empty and unloaded cluster. The system tries to find at least one queue in which the job could run. Possible checking modes are as follows:
  - Skip - No consistency checking at all.
  - Warning - Inconsistencies are reported, but the job is still accepted. Warning mode might be desirable if the cluster configuration should change after the job is submitted.
  - Error - Inconsistencies are reported. The job is rejected if any inconsistencies are encountered.
  - Just verify - The job is not submitted. An extensive report is generated about the suitability of the job for each host and queue in an empty cluster.
  - Poke - The job is not submitted. An extensive report is generated about the suitability of the job for each host and queue in the cluster with all resource utilizations in place.
- Advance Reservation - A list of available, configured advance reservations.
- JSV URL - Access to your directory to select from configured server JSV scripts.
- Mail - The events about which the user is notified by email. The events' start, end, abort, and suspend are currently defined for jobs.
- Mail To - A list of email addresses to which these notifications are sent. Click the icon at the right of the Mail To field to open a dialog box for defining the mailing list.
- Hard Queue List, Soft Queue List - A list of queue names that are requested to be the mandatory selection for the execution of the job. The Hard Queue List and the Soft Queue List are treated identically to a corresponding resource requirement.
- Master Queue List - A list of queue names that are eligible as master queue for a parallel job. A parallel job is started in the master queue. All other queues to which the job spawns parallel tasks are called slave queues.
- Job Dependencies - A list of IDs of jobs that must finish before the submitted job can be started. The newly created job depends on completion of those jobs.
- Hold Array Dependencies - A list of job IDs/and/or job names and sub-tasks. Each sub-task of the submitted job is not eligible for execution unless the corresponding sub-tasks of all jobs referenced in the comma-separated job ID and/or job name list have completed.
- Deadline - The deadline initiation time for deadline jobs. Deadline initiation defines the point in time at which a deadline job must reach maximum priority to finish before a given deadline. To determine the deadline initiation time, subtract an estimate of the running time, at maximum priority, of a

deadline job from its desired deadline time. Click the icon at the right of the Deadline field to open the dialog box that enables you to set the deadline.

Note: Not all users are allowed to submit deadline jobs. Ask your system administrator if you are permitted to submit deadline jobs. Contact the cluster administrator for information about the maximum priority that is given to deadline jobs.

2.16 How to Configure Job Dependencies From the Command Line

Often the most convenient way to build a complex task is to split the task into subtasks. The Grid Engine system supports interdependent tasks with its job dependency facility. In these cases, subtasks depend on the completion of other subtasks before the dependent subtasks can get started. An example is that a predecessor task produces an output file that must be read and processed by a dependent task.

You can configure jobs to depend on the completion of one or more other jobs. The facility is enforced by the `qsub -hold_jid` command. You can specify a list of jobs upon which the submitted job depends. The list of jobs can also contain subsets of array jobs. The submitted job is not eligible for execution unless all jobs in the dependency list have finished.

To hold a job until another job has finished, type the following command:

```
qsub -hold_jid <jobid>
```

You can specify a list of jobs upon which the submitted job depends. The list of jobs can also contain subsets of array jobs. The submitted job is not eligible for execution unless all jobs in the dependency list have finished.

`qsub -hold_jid 45,46,47,48,49,50` scriptname would ensure that script name wasn't run until after all the jobs numbered 45 to 50 had completed.

2.17 Monitoring Hosts from the Command Line

Learn how to monitor hosts from the command line.

2.17.1 Using qconf

To display an execution host configuration, type the following command:

```
% qconf -se <hostname>
```

The `-se` option (show execution host) shows the configuration of the specified execution host as defined in `host_conf`.

To display an execution host list, type one of the following command:

```
% qconf -sel
```

The `-sel` option (show execution host list) displays a list of hosts that are configured as execution hosts.

2.17.2 Using qhost

To monitor execution hosts from the command line, type the following command:

```
% qhost
```

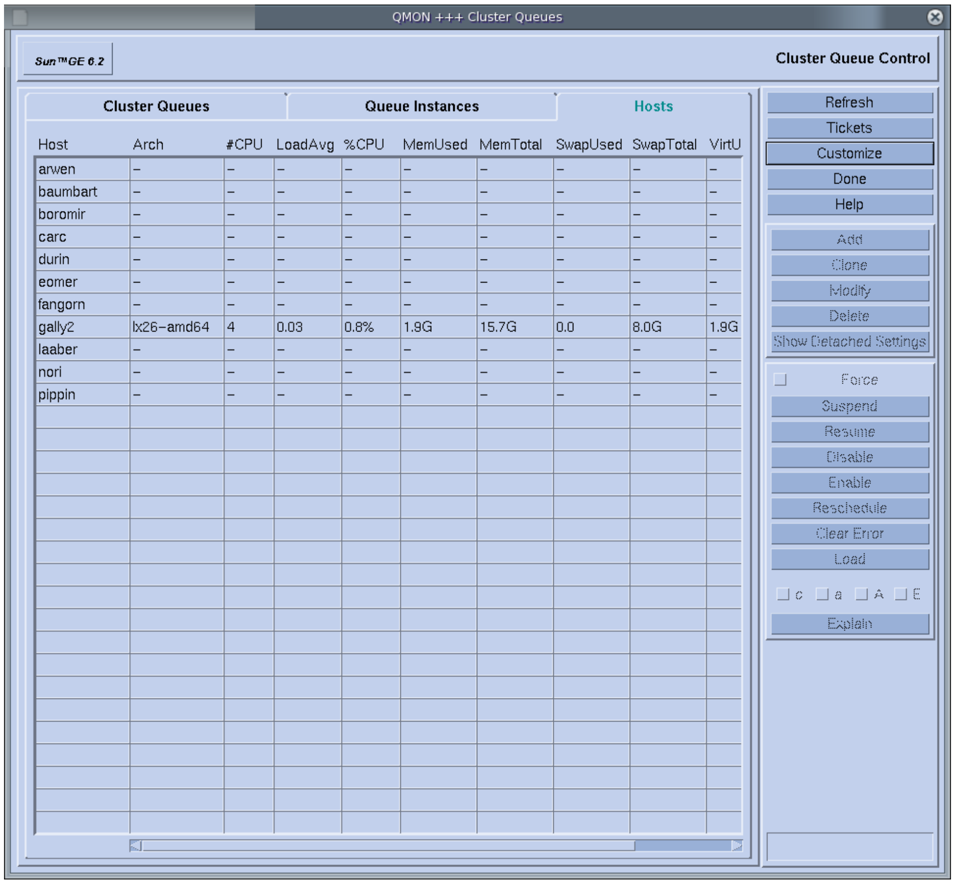
This command produces output that is similar to the following example:

```
HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS
-------------------------------------------------------------------------------
global - - - - - - -
grid1 sol-sparc64 2 0.27 2.0G 256.0M 8.0G 0.0
gridengine2 sol-amd64 4 0.00 3.9G 421.0M 2.0G 0.0
gridengine5 sol-amd64 4 0.00 3.9G 488.0M 7.9G 0.0
gridengine6 sol-amd64 4 0.07 3.9G 2.6G 4.0G 0.0
```

2.18 How to Monitor Hosts With QMON

1. Click the Queue Control button in the QMON Main Control window. The Cluster Queues dialog box appears.
2. Click the Hosts tab. The Hosts tab provides a quick overview of all hosts that are available for the cluster.

Figure 2–10 QMON Cluster Queues



Host	Arch	#CPU	LoadAvg	%CPU	MemUsed	MemTotal	SwapUsed	SwapTotal	VirtU
arwen	–	–	–	–	–	–	–	–	–
baumbart	–	–	–	–	–	–	–	–	–
boromir	–	–	–	–	–	–	–	–	–
carc	–	–	–	–	–	–	–	–	–
durin	–	–	–	–	–	–	–	–	–
eomer	–	–	–	–	–	–	–	–	–
fangorn	–	–	–	–	–	–	–	–	–
gally2	lx26–amd64	4	0.03	0.8%	1.9G	15.7G	0.0	8.0G	1.9G
laaber	–	–	–	–	–	–	–	–	–
nori	–	–	–	–	–	–	–	–	–
pippin	–	–	–	–	–	–	–	–	–

2.18.1 Hosts Status

Each row in the hosts table represents one host. For each host, the table lists the following information:

- Host - Name of the host

- Arch - Architecture of the host
- #CPU - Number of processors
- LoadAvg - Load average of the host
- %CPU - LoadAvg/(#CPU * 100)
- MemUsed - Used Memory
- MemTotal - Total Memory
- SwapUsed - Used Swap Memory
- SwapTotal - Total Swap Memory
- VirtUsed - Virtual Used Memory
- VirtTotal - Virtual Total Memory

2.19 Monitoring and Controlling Jobs

After you submit jobs, you need to monitor and control them. The following page provides information about monitoring and controlling jobs.

Note: Only the job owner or Grid Engine managers and operators can suspend and resume jobs, delete jobs, hold back jobs, modify job priority, and modify attributes. See Displaying User Properties.

2.19.1 How to Monitor Jobs From the Command Line

Use the `qstat` command to perform the following monitoring functions:

- To display a list of jobs with no queue status information, type the following command:

  ```
  qstat
  ```

  The purpose of most of the columns should be self-explanatory. The state column, however, contains single character codes with the following meaning: `r` for running, `s` for suspended, `q` for queued, and `w` for waiting.

- To display summary information on all queues and the queued job list, type the following command:

  ```
  qstat -f
  ```

  The display is divided into the following two sections:

  - Available Queues - This section displays the status of all available queues. The first line of the queue section defines the meaning of the columns with respect to the queues that are listed. The queues are separated by horizontal lines. If jobs run in a queue, the job names appear below the associated queue in the same format as in the `qstat` command in its first form. The columns of the queue description provide the following information:

    * `qtype` - Queue type. Queue type is either `B` (batch) or `I` (interactive).

    * `used/free` - Count of used and free job slots in the queue.

    * `states` - State of the queue.

- Pending Jobs - This section shows the status of the `sge_qmaster` job spool area. The pending jobs in the second output section are also listed as in qstat`s first form.

- To display current job usage and ticket information for a job, type the following command:

  ```
  qstat -ext
  ```

  This command contains details such as up-to-date job usage and tickets assigned to a job. The following information is displayed:

  - The usage and ticket values assigned to a job, shown in the following columns:
    - `cpu/mem/io` - Currently accumulated CPU, memory, and I/O usage.
    - `tckts` - Total number of tickets assigned to the job.
    - `ovrts` - Override tickets assigned through `qalter -ot`.
    - `otckt` - Tickets assigned through the override policy.
    - `ftckt` - Tickets assigned through the functional policy.
    - `stckt` - Tickets assigned through the share-based policy.
    - `Share` - Current Resource share that each job has with respect to the usage generated by all jobs in the cluster.
  - The deadline initiation time in the column deadline, if applicable.

Additional options to the `qstat` command enhance the functionality. Use the `-r` option to display the resource requirements of submitted jobs. Furthermore, the output can be restricted to a certain user or to a specific queue. You can use the `-l` option to specify resource requirements, as described in Defining Resource Requirements, for the `qsub` command. If resource requirements are used, only those queues, and the jobs that are running in those queues, are displayed that match the resource requirement specified by `qstat`.

Note: The `qstat` command has been enhanced so that the administrator and the user may define files that can contain useful options. A cluster-wide `sge_qstat` file may be placed under `$xxQS_NAME_Sxx_ROOT/$xxQS_NAME_Sxx_CELL/common/sge_qstat`. The user private file is processed under the location `$HOME/.sge_qstat`. The home directory request file has the highest precedence, then the cluster global file. You can use the command line to override the flags contained in a file.

The following examples show output from the `qstat` and `qstat -f` commands.

Example - qstat -f Output

```
queuename                  qtype  used/free   load_avg   arch         states

dq                         BIP    0/1         99.99      sun4         au

durin.q                    BIP    2/2         0.36       sun4
  231     0    hydra              craig       r          07/13/96     20:27:15
MASTER
  232     0    compile            penny       r          07/13/96     20:30:40
MASTER
```

```
dwain.q                  BIP    3/3        0.36       sun4
  230     0    blackhole          don        r          07/13/96     20:26:10
MASTER
  233     0    mac                elaine     r          07/13/96     20:30:40
MASTER
  234     0    golf               shannon    r          07/13/96     20:31:44
MASTER

fq                       BIP    0/3        0.36       sun4

################################################################################

- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS -

################################################################################

  236     5    word               elaine     qw         07/13/96     20:32:07

  235     0    andrun             penny      qw         07/13/96     20:31:43
```

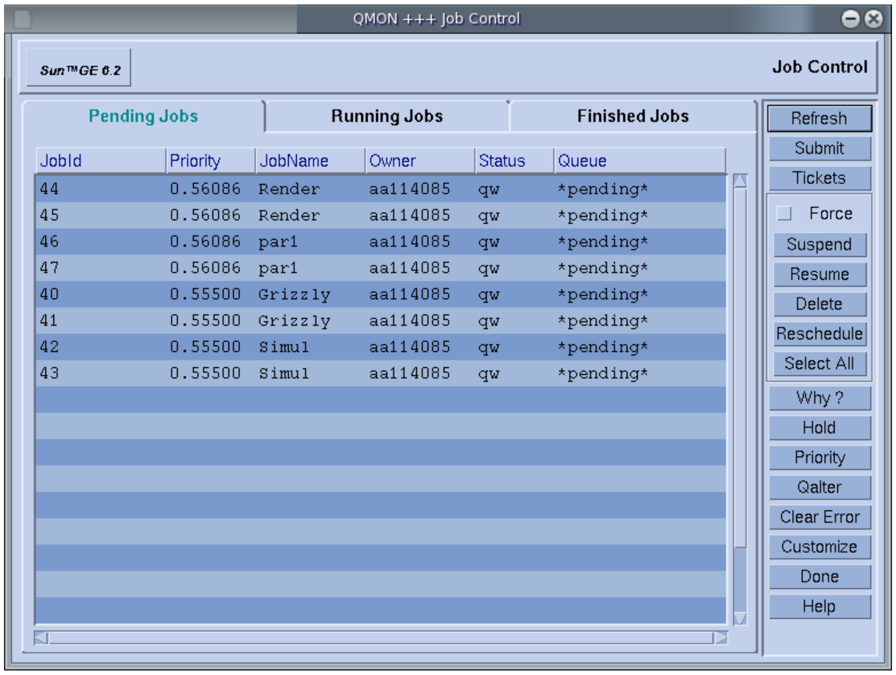
Example - qstat Output

```
job-ID   prior   name         user      state   submit/start at     queue
function
231      0       hydra        craig     r       07/13/96            durin.q
MASTER
                                                20:27:15
232      0       compile      penny     r       07/13/96            durin.q
MASTER
                                                20:30:40
230      0       blackhole    don       r       07/13/96            dwain.q
MASTER
                                                20:26:10
233      0       mac          elaine    r       07/13/96            dwain.q
MASTER
                                                20:30:40
234      0       golf         shannon   r       07/13/96            dwain.q
MASTER
                                                20:31:44
236      5       word         elaine    qw      07/13/96
                                                20:32:07
235      0       andrun       penny     qw      07/13/96 20:31:43
```

2.19.2 How to Monitor Jobs With QMON

To monitor jobs with QMON, click the Job Control button in the QMON Main Control window. The Job Control dialog box appears, as shown below.

Figure 2–11 QMON Job Control



JobId	Priority	JobName	Owner	Status	Queue
44	0.56086	Render	aa114085	qw	*pending*
45	0.56086	Render	aa114085	qw	*pending*
46	0.56086	par1	aa114085	qw	*pending*
47	0.56086	par1	aa114085	qw	*pending*
40	0.55500	Grizzly	aa114085	qw	*pending*
41	0.55500	Grizzly	aa114085	qw	*pending*
42	0.55500	Simul	aa114085	qw	*pending*
43	0.55500	Simul	aa114085	qw	*pending*

2.19.2.1 How to Get Additional Information About Jobs With the QMON Object Browser

You can use the QMON Object Browser to quickly retrieve additional information about jobs without having to customize the Job Control dialog box, as explained in How to Monitor Jobs With QMON.

To display information about jobs using the Object Browser, use one of the following methods:

- From the Job Control dialog box, move the pointer over a job name.
- From the Browser dialog box, click Job.

2.19.3 How to Control Jobs From the Command Line

Note: In order to delete, suspend, or resume a job, you must be the owner of the job or a Grid Engine manager or operator. For more information, see Users and User Categories.

Use `qdel` and `qmod` in the following ways to control jobs from the command line:

- To delete a job, regardless of whether a job is running or spooled, type the following command:

  ```
  qdel <job-id>
  ```

- To suspend a job that is already running, type the following command:

  ```
  qmod -sj <job-id>
  ```

- To restart a suspended job, type the following command:

  ```
  qmod -usj <job-id>
  ```

To retrieve a `job_id` number, use `qstat`. For more information, see How to Monitor Jobs From the Command Line.

If an execution daemon is unreachable, you can use the `-f` (force) option with both commands to register a job status change at master daemon. The `-f` option is intended for use only by an administrator. However, In the case of `qdel`, users can force deletion of their own jobs if the flag `ENABLE_FORCED_QDEL` in the cluster configuration `qmaster_params` entry is set.

2.19.4 How to Control Jobs With QMON

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears, as shown below.
2. You can perform the following tasks from the Job Control dialog box:

   Note: To select jobs, use the following mouse and key combinations:

   - To select multiple noncontiguous jobs, hold down the Control key and click two or more jobs.
   - To select a contiguous range of jobs, hold down the Shift key, click the first job in the range, and then click the last job in the range.
   - To toggle between selecting a job and clearing the selection, click the job while holding down the Control key.

   - To monitor jobs, click the Pending Jobs, Running Jobs, or Finished Jobs tab.
   - To refresh the Job Control display, click the Refresh button to force an update. QMON then uses a polling scheme to retrieve the status of the jobs from `sge_qmaster`.
   - To modify job attributes, select a pending or running job and then click the Qalter button. For more information, see How to Modify Job Attributes.
   - To change job priority, select a pending or running job and then press the Priority button. For more information, see How to Change Job Priority.
   - To put a job or an array task on hold, select a pending job and then press the Hold button. For more information, see How to Put Jobs and Array Job Tasks on Hold.
   - To force a job, first select a pending job or running job, next select the Force option and then click the Suspend, Resume, or Delete buttons. For more information, see How to Force Jobs.
   - To verify job consistency, select a pending job and then click the Qalter button. For more information, see How to Verify Job Consistency.
   - To get information about pending jobs using the Why? button, select a pending job and then click the Why? button. For more information, see How to Use the Why? Button to Get Information About Pending Jobs.
   - To clear error states, select a pending job and then click the Clear Error button. For more information, see How to Clear Error States.
   - To filter the job list, click the Customize button. For more information, see How to Filter the Job List.

- To customize the job control display, click the Customize button. For more information, see How to Customize the Job Control Display.

How to Modify Job Attributes

1. Click a pending or running job on the and Job Control dialog box and then click the Qalter button. The Submit Job dialog box appears. All the entries of the dialog box correspond to the attributes of the job that were defined when the job was submitted.

   Note: Entries that cannot be changed are grayed out.

2. Edit available entries appropriately.
3. Click the Qalter button, a substitute for the Submit button on the Submit Job dialog box, to register changes with the Grid Engine system.

How to Change Job Priority

1. Select a pending or running job on the Job Control dialog box and then click the Priority button. The priority dialog box appears, as shown below. This dialog box enables you to change the priority of selected pending or running jobs. The priority ranks a single user's jobs among themselves. Priority tells the scheduler how to choose among a single user's jobs when several jobs are in the system simultaneously.
2. Enter a new priority for the selected job(s) in the field and then click OK.

How to Put Jobs and Array Job Tasks on Hold

As long as any hold is assigned to a job or an array job task, the job or array job task is not eligible for running.

Note: User holds can be set or reset by the job owner as well as by Grid Engine managers and operators. Operator holds can be set or reset by managers and operators. System holds can be set or reset by managers only. You can also set or reset holds by using the `qalter`, `qhold`, and `qrls` commands.

- To put a job on hold, select a pending job from the Job Control Dialog dialog box, shown above, and click Hold. The Set Hold dialog box appears. The Set Hold dialog box enables you to set and reset user, operator, and system holds.
- To put an array task on hold, do the following:
  1. Select a pending job from the Job Control dialog box and click Hold. The Set Hold dialog box appears.
  2. Use the Tasks field to put a hold on particular subtasks of an array job. The task ID range specified in this field can be a single number, a simple range of the form `n-m`, or a range with a step size. For example, the task ID range specified by `2-10:2` results in the task ID indexes 2, 4, 6, 8, and 10. This range represents a total of five identical tasks, with the environment variable `SGE_TASK_ID` containing one of the five index numbers.

How to Force Jobs

Only running jobs can be suspended or resumed. Only pending jobs can be rescheduled, held back and modified, in priority as well as in other attributes.

1. To force jobs, select a job from the Pending Jobs tab or the Running Jobs tab and then select the Force option.
2. Click the Suspend, Resume, or Delete buttons.

Note: You can force suspending, resuming, and deleting jobs. In other words, you can register these actions with `sge_qmaster` without notifying the `sge_execd` that controls the jobs. Forcing is useful when the corresponding `sge_execd` is unreachable, for example, due to network problems.

Suspension of a job sends the signal `SIGSTOP` to the process group of the job with the UNIX `kill` command. `SIGSTOP` halts the job and no longer consumes CPU time. Resumption of the job sends the signal `SIGCONT`, thereby unsuspending the job.

How to Verify Job Consistency

Note: The Verify flag on the Submit Job dialog box has a special meaning when the flag is used in the Qalter mode. You can check pending jobs for consistency, and you can investigate why jobs are not yet scheduled.

---

1. Select a pending job from the Job Control dialog box and click the Qalter button.
2. Click the Advanced tab.
3. Select the desired consistency-checking mode for the Verify flag, and then click Qalter.

Note: The system displays warnings on inconsistencies, depending on the checking mode you select. See How to Submit an Advanced Job With QMON for more information.

How to Use the Why? Button to Get Information About Pending Jobs

---

Note: The Why? button delivers meaningful output only if the scheduler configuration parameter `schedd_job_info` is set to true.

To get information about pending jobs, select a pending job from the Job Control dialog box and click the Why? button. The Object Browser dialog box appears. As shown below, this dialog box displays a list of reasons that prevented the scheduler from dispatching the job in its most recent pass.

Note: The displayed scheduler information relates to the last scheduling interval. The information might not be accurate by the time you investigate why your job was not scheduled.

How to Clear Error States

To clear error states, select a pending job from the Job Control dialog box and then click the Clear Error button. This removes an error state from a pending job that failed due to a job-dependent problem. For example, the job might have insufficient permissions to write to the specified job output file.

Error states appear in red text in the pending jobs list. You should remove jobs only after you correct the error condition, for example, using `qalter`. Such error conditions are automatically reported through email if the job requests to send email when the job is aborted. For example, the job might have been aborted with the `qsub -m` a command.

How to Filter the Job List

1. Click the Customize button in the Job Control dialog box. The Job Customize box appears, as shown below.
2. Click the Filter Jobs tab.

Example - Filtering the Job List

The following example of the filtering facility selects only jobs that are suitable to be run on the architecture solaris64. The following figure shows the resulting Running Jobs tab of the Job Control dialog box. The Job Control dialog box that is shown in the previous figure is also an example of how QMON displays array jobs.

How to Customize the Job Control Display

1. Click the Customize button on the Job Control dialog box. The Job Customize dialog box appears.
2. Click the Select Job Fields tab. A sample Select Job Fields tab is shown in the following figure.
3. Use the Job Customize dialog box to configure the set of information to display. You can select more entries of the job object to be displayed.
4. Use the Save button on the Job Customize dialog box to store the customizations in the file `.qmon_preferences`. This file is located in the user's home directory. By saving your customizations, you redefine the appearance of the Job Control dialog box.

2.19.5 How to Monitor Jobs by Email

From the command line, type the following command with appropriate arguments.

```
% qsub -m <arguments>
```

The `qsub -m` command requests email to be sent to the user who submitted a job or to the email addresses specified by the `-M` flag if certain events occur. An argument to the `-m` option specifies the events. The following arguments are available:

- `b` - Send email at the beginning of the job.
- `e` - Send email at the end of the job.
- `a` - Send email when the job is rescheduled or aborted For example, by using the `qdel` command.
- `s` - Send email when the job is suspended.
- `n` - Do not send `email.n` is the default.

Use a string made up of one or more of the letter arguments to specify several of these options with a single `-m` option. For example, `-m` be sends email at the beginning and at the end of a job.

2.19.6 How to Monitor Jobs by Email With QMON

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select a pending job and click the Qalter button. The Submit Job dialog box appears, as shown below.
3. Select the Advanced Tab.
4. Click on the icon left of the Mail To field to select or add email addresses of the user or users who are responsible for monitoring jobs.

> **Note:** You can also configure this parameter at the time of job submission using the Submit Job dialog box.

2.20 Monitoring and Controlling Queues

After you configure queues, you need to monitor and control them. This page provides information about monitoring and controlling queues.

2.20.1 How to Control Queues From the Command Line

> **Note:** Suspending and resuming queues as well as disabling and enabling queues requires queue owner permission, manager permission, or operator permission. For more information, see Users and User Categories.

You can use `qmod` to control queues in the following ways:

- To suspend a queue and any active jobs on that queue, type the following command:

  ```
  qmod -sq <q-name>,...
  ```

- To unsuspend a queue and any active jobs on that queue, type the following command:

  ```
  qmod -usq <q-name>,...
  ```

- To disable a queue and stop any jobs from being dispatched to the queue, type the following command:

  ```
  qmod -d <q-name>,...
  ```

- To enable a queue, type the following command:

  ```
  qmod -e <q-name>,...
  ```

The `-f` option forces registration of the status change in `sge_qmaster` when the corresponding `sge_execd` is not reachable, for example, due to network problems.

2.20.2 How to Monitor and Control Cluster Queues With QMON

1. Click the Queue Control button in the QMON Main Control window. The Cluster Queues dialog box appears, as shown below.
2. Click the Cluster Queues tab. The Cluster Queues tab provides a quick overview of all cluster queues that are defined for the cluster.

   Note: Information displayed in the Cluster Queues dialog box is updated periodically. Click Refresh to force an update.

3. Select a cluster queue name.
4. Click Delete, Suspend, Resume, Disable, or Enable to execute the corresponding operation on cluster queues that you select.

   Note: The suspend/resume and disable/enable operations require notification of the corresponding `sge_execd`. If notification is not possible, you can force an `sge_qmaster` internal status change by clicking Force. For example, notification might not be possible because a host is down.

   The suspend/resume and disable/enable operations require cluster queue owner permission, Grid Engine manager permission, or operator permission. See Users and User Categories for details.

   Suspended cluster queues are closed for further jobs. The jobs already running in suspended queues are also suspended, as described in Monitoring and Controlling Jobs. The cluster queue and its jobs are unsuspended as soon as the queue is resumed.

   Note: If a job in a suspended cluster queue was suspended explicitly, the job is not resumed when the queue is resumed. The job must be resumed explicitly. Disabled cluster queues are closed. However, the jobs that are running in those queues are allowed to continue. The disabling of a cluster queue is commonly used to clear a queue. After the cluster queue is enabled, it is eligible to run jobs again. No action on currently running jobs is performed.

5. Click Clear Error to remove an error state from a queue. Error states are displayed using a red font in the queue list.
6. Click Reschedule to reschedule all jobs currently running in the selected cluster queues.
7. Click Add or Modify on the Cluster Queue dialog box to configure cluster queues and queue instances. See *Oracle Grid Engine Administration Guide* for configuring queues details.
8. Click Done to close the dialog box.

2.20.2.1 Cluster Queue Status

Each row in the cluster queue table represents one cluster queue. For each cluster queue, the table lists the following information:

- Cluster Queue - Name of the cluster queue.
- Load - Average of the normalized load average of all cluster queue hosts. Only hosts with a load value are considered.
- Used - Number of currently used job slots.
- Avail - Number of currently available job slots.
- Total - Total number of job slots.
- aoACD - Number of queue instances that are in at least one of the following states:
  - a - Load threshold alarm
  - o - Orphaned
  - A - Suspend threshold alarm
  - C - Suspended by calendar
  - D - Disabled by calendar
- cdsuE - Number of queue instances that are in at least one of the following states:
  - c - Configuration ambiguous
  - d - Disabled
  - s - Suspended
  - u - Unknown
  - E - Error
- s - Number of queue instances that are in the suspended state.
- A - Number of queue instances where one or more suspend thresholds are currently exceeded. No more jobs
- S - Number of queue instances that are suspended through subordination to another queue.
- C - Number of queue instances that are automatically suspended by the Grid Engine system calendar.
- u - Number of queue instances that are in an unknown state.
- a - Number of queue instances where one or more load thresholds are currently exceeded.
- d - Number of queue instances that are in the disabled state.
- D - Number of queue instances that are automatically disabled by the Grid Engine system calendar.
- c - Number of queue instances whose configuration is ambiguous.
- o - Number of queue instances that are in the orphaned state.
- E - Number of queue instances that are in the error state.

2.20.3 How to Monitor Queues With QMON

1. Click the Queue Control button in the QMON Main Control window. The Cluster Queues dialog box appears, as shown below.
2. Click the Cluster Queues tab. The Cluster Queues tab provides a quick overview of all cluster queues that are defined for the cluster.

Note: Information displayed in the Cluster Queues dialog box is updated periodically. Click Refresh to force an update.

2.21 Using Job Checkpointing

For an introduction to checkpointing and checkpointing environments, see *Oracle Grid Engine Administration Guide* for managing checkpointing environments.

2.21.1 Migrating Checkpointing Jobs

Checkpointing jobs are interruptible at any time because their restart capability ensures that very little work that is already done must be repeated. This ability is used to build migration and dynamic load balancing mechanism in the Grid Engine system. If requested, checkpointing jobs are stopped on demand. The jobs are migrated to other machines in the Grid Engine system, thus averaging the load in the cluster dynamically. Checkpointing jobs are stopped and migrated for the following reasons:

- The executing queue or the job is suspended explicitly by a qmod or a QMON command.
- The job or the queue where the job runs is suspended automatically because a suspend threshold for the queue is exceeded. The checkpoint occasion specification for the job includes the suspension case. `For more information, see How to Configure Load and Suspend Thresholds and How to Submit, Monitor, or Delete a Checkpointing Job`.

A migrating job moves back to `sge_qmaster`. The job is subsequently dispatched to another suitable queue if such a queue is available. In such a case, the `qstat` output shows `R` as the status.

2.21.2 File System Requirements for Checkpointing

When a user-level checkpoint or a kernel-level checkpoint that is based on a checkpointing library is written, a complete image of the virtual memory covered by the process or job to be checkpointed must be saved. Sufficient disk space must be available for this purpose. If the checkpointing environment configuration parameter `ckpt_dir` is set, the checkpoint information is saved to a job private location under `ckpt_dir`. If `ckpt_dir` is set to `NONE`, the directory where the checkpointing job started is used.

Note: You should start a checkpointing job with the `qsub -cwd` script if `ckpt_dir` is set to `NONE`.

Checkpointing files and restart files must be visible on all machines in order to successfully migrate and restart jobs. Because file visibility is necessary for the way file systems must be organized, NFS or a similar file system is required. Ask your cluster administration if your site meets this requirement.

If your site does not run NFS, you can transfer the restart files explicitly at the beginning of your shell script. For example, you can use `rcp` or `ftp` in the case of user-level checkpointing jobs.

2.21.3 Writing a Checkpointing Job Script

Shell scripts for kernel-level checkpointing are the same as regular shell scripts.

Shell scripts for user-level checkpointing jobs differ from regular batch scripts only in their ability to properly handle the restart process. The environment variable RESTARTED is set for checkpointing jobs that are restarted. Use this variable to skip sections of the job script that need to be executed only during the initial invocation.

The following example shows a sample transparently checkpointing job script.

Example – Checkpointing Job Script

```
#!/bin/sh
#Force /bin/sh in Grid Engine
#$ -S /bin/sh

# Test if restarted/migrated
if [ $RESTARTED = 0 ]; then
     # 0 = not restarted
     # Parts to be executed only during the first
     # start go in here
     set_up_grid
fi

# Start the checkpointing executable
fem
#End of scriptfile
```

The job script restarts from the beginning if a user-level checkpointing job is migrated. The user is responsible for directing the program flow of the shell script to the location where the job was interrupted. Doing so skips those lines in the script that must be executed more than once.

> **Note:** Kernel-level checkpointing jobs are interruptible at any time. The embracing shell script is restarted exactly from the point where the last checkpoint occurred. Therefore, the RESTARTED environment variable is not relevant for kernel-level checkpointing jobs.

2.21.4 How to Submit a Checkpointing Job From the Command Line

Checkpointing job scripts are similar to regular batch scripts with the exception of the `qsub -ckpt` and `qsub -c` commands. These commands request a checkpointing mechanism define the occasions at which checkpoints must be generated for the job.

- To select a checkpointing environment for a job, use the following argument for the `qsub` command:

  ```
  qsub -ckpt <ckpt_name>
  ```

  This argument is also available for qalter.

- To define (or redefine) how a job should be checkpointed, use the following commmand:

  ```
  qsub -c <occasion_specifier>
  ```

  The `-c` option is not required. This argument can be used to overwrite the definitions of the when parameter in the checkpointing environment that is referenced by the `qsub -ckpt` switch. This argument is also available for `qalter`.

For information on configuring checkpointing environments, see *Oracle Grid Engine Administration Guide* for managing checkpointing environments.

2.21.5 How to Submit a Checkpointing Job With QMON

The submission of checkpointing jobs With QMON is identical to submitting regular batch jobs, with the addition of specifying an appropriate checkpointing environment.

1. Click the Job Control button in the QMON Main Control window. The Job Control dialog box appears.
2. Select a pending job and click the Qalter button. The Submit Job dialog box appears.
3. Click the Advanced Tab, which shown below.
4. Click the button next to that field to open the following Selection dialog box.
5. Select a suitable checkpointing environment from the list of available checkpoint objects. Ask your system administrator for information about the properties of the checkpointing environments that are installed at your site.

For more information, see *Oracle Grid Engine Administration Guide* for managing checkpointing environments.

2.22 Managing Core Binding

1. PROCEDURE MISSING

2.22.1 Submit Simple Jobs with Core Binding

You can submit simple jobs with core binding. The following example tries to bind the binary sleep on two successive cores when possible on a single socket. Additionally an execution host is requested that has 4 cores.

```
qsub -b y -binding linear:2 -l m_core=4 sleep 3600
```

2.22.2 Submit Array Jobs with Core Binding

You can use core binding with array jobs but this is not recommended for the explicit request, and the linear, and striding with a given start point. In these cases, only one solution for binding is valid and therefore just the first task can be bound.

In the following example, eight array tasks each running on a different core, are spawned.

```
% qsub -b y -t 1:8 -binding linear:1 -l m_core=8 sleep 3600
```

On an eight core host with Solaris operating system the following binding is done:

```
% qstat -cb -j <jobno>

...
job_args:                   3600
script_file:                sleep
job-array tasks:            1-8:1
binding:                    set linear:1
...
binding    1:               ScttCTTCTTCTTSCTTCTTCTTCTT
binding    2:               SCTTCTTCTTCTTScttCTTCTTCTT
binding    3:               SCTTcttCTTCTTSCTTCTTCTTCTT
binding    4:               SCTTCTTCTTCTTSCTTcttCTTCTT
binding    5:               SCTTCTTcttCTTSCTTCTTCTTCTT
binding    6:               SCTTCTTCTTCTTSCTTCTTcttCTT
binding    7:              SCTTCTTCTTcttSCTTCTTCTTCTT
binding    8:              NONE
...
```

More detailed information regarding the task eight not being bound, see the processor set feature of Solaris OS. As all other seven jobs have one core exclusively bound to them, the last remaining core can not be used for core binding as the operating system itself have to be run somewhere. But in this case, the task eight is running on the remaining core.

In this example, the core allocation scheme of the linear request can be examined. The first core is allocated to the socket which is free. Therefore, the second task is running on the second socket. For the third task, no sockets are free hence the first socket with the most free cores is used. Therefore job four can be found on the other socket and so on.

When submitting the same job with the striding strategy on a Linux operating system the output is slightly different. Here, each task is bound and each task takes the first free core that can be used.

```
% qsub -b y -binding striding:1:1 -l m_core=8 sleep 3600

% qstat -cb -j <jobid>
....
job_args:                  3600
script_file:               sleep
job-array tasks:           1-8:1
binding:                   set striding:1:1
...
binding    1:              ScttCTTCTTCTTSCTTCTTCTTCTT
binding    2:              SCTTcttCTTCTTSCTTCTTCTTCTT
binding    3:              SCTTCTTcttCTTSCTTCTTCTTCTT
binding    4:              SCTTCTTCTTcttSCTTCTTCTTCTT
binding    5:              SCTTCTTCTTCTTScttCTTCTTCTT
binding    6:              SCTTCTTCTTCTTSCTTcttCTTCTT
binding    7:              SCTTCTTCTTCTTSCTTCTTcttCTT
binding    8:              SCTTCTTCTTCTTSCTTCTTCTTctt
...
```

2.22.3 Submit Parallel Jobs with Core Binding

You can submit parallel jobs with core binding. The recommended way of starting parallel/multi-threaded jobs is to use a parallel environment. The number of requested cores is reflected with the slots variable. In the following example, the parallel environment `mytestpe` is used.

```
% qconf -sp mytestpe
pe_name            mytestpe
slots              1024
user_lists         NONE
xuser_lists        NONE
start_proc_args    NONE
stop_proc_args     NONE
allocation_rule    $pe_slots
control_slaves     FALSE
job_is_first_task  TRUE
urgency_slots      min
accounting_summary FALSE
```

In the example below, a four way parallel loosely integrated job is started and is running on one host (because the allocation rule `$pe_slots` that was configured in the parallel environment `mytestpe`).

```
% qsub -b y -pe mytestpe 4 -binding linear:4 sleep 3600
```

Multi-process or multi-threaded array tasks can be submitted in the same way. The following examples shows the submission of four array tasks each using two slots.

```
% qsub -b y -pe mytestpe 2 -t 1:4 -binding linear:2 sleep 3600
```

2.22.3.1 Submit Tightly Integrated Parallel Jobs with Core Binding

When you submit parallel jobs, the same limitations as for array tasks is taken into account. It means that only the linear and striding strategy without a given start point make sense. Otherwise, just one task per host will be bound because of core collisions.

In case of tight integration, the `qrsh --inherit` does not support the binding request as parameter. The request is available on the execution host for the tasks. It means when `-binding linear:1` was requested (via `qsub`) each task is started with `qrsh` and tried to be bound on a different core. Hence, when the connection is established the job run will be bound according the request.

2.23 Automating Grid Engine Functions Through DRMAA

You can automate Grid Engine functions by writing scripts that run Grid Engine commands and parse the results. However, for more consistent and efficient results, you can use the C or Java language and the Distributed Resource Management Application API. This section introduces the DRMAA concept and explains how to use it with the C and Java languages.

The Distributed Resource Management Application API (DRMAA, which is pronounced like "drama") is an Open Grid Forum specification to standardize job submission, monitoring, and control in Distributed Resource Management Systems (DRMS). The objective of the DRMAA Working Group was to produce an API that would be easy to learn, easy to implement, and that would enable useful application integrations with DRMS in a standard way.

The DRMAA specification is language, platform, and DRMS agnostic. A wide variety of systems should be able to implement the DRMAA specification. To provide additional guidance for DRMAA implementation in specific languages, the DRMAA Working Group also produced several DRMAA language binding specifications. These specifications define what a DRMAA implementation should resemble in a given language.

The DRMAA specification is currently at version 1.0. The DRMAA Java Language Binding Specification is also at version 1.0, as is the DRMAA C Language Binding Specification. Grid Engine provides implementations of both the 1.0 Java language binding and the 1.0 C language binding.

For more information about the DRMAA 1.0 specification, see the language specific binding specifications on the Open Grid Forum DRMAA Working Group Web Site

2.23.1 Developing With the C Language Binding

2.23.1.1 Important Files for the C Language Binding

To use the DRMAA C language binding implementation included with Grid Engine, you need to know where to find the important files. The most important file is the DRMAA header file that you included from your C application to make the DRMAA functions available to your application. The DRMAA header file resides in the `$SGE_ROOT/include/drmaa.h`, where `$SGE_ROOT` defaults to `/usr/SGE`. To compile and link your application, use the DRMAA shared library at `$SGE_ROOT/lib/$SGE_ARCH/libdrmaa.so`.

2.23.1.2 Including the DRMAA Header File

To use the DRMAA functions in your application, every source file that uses a DRMAA function must include the DRMAA header file. To include the DRMAA header file in your source file, add the following line to your source code:

```
#include "drmaa.h"
```

2.23.1.3 Compiling Your C Application

When you compile your DRMAA application, you need to include some additional compiler directives to direct the compiler and linker to use DRMAA. The following directions apply to the Sun Studio Compiler Collection and to `gcc`. These instructions might not apply for other compilers and linkers. Consult the documentation for your specific compiler and linker products.

You must include the following two directives:

- Tell the compiler to include the DRMAA header file by adding the following statement to the compiler command line:

  ```
  -$SGE_ROOT/include
  ```

- Tell the linker to include the DRMAA library by adding the following statement to the compiler and/or linker command line:

  ```
  -ldrmaa
  ```

You also need to verify that the `$SGE_ROOT/lib/$SGE_ARCH` directory is included in your library search path. The path is `LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux. The `$SGE_ROOT/lib/$SGE_ARCH` directory is not included automatically when you set your environment using the `settings.sh` or `settings.csh` files.

Example - Compiling Your C Application Using Sun Studio Compiler

The following example shows how you would compile your DRMAA application using the Sun Studio Compiler. The following assumptions apply:

- You are using the `csh` shell on a Solaris host.
- Grid Engine is installed in `/sge`.
- The DRMAA application is stored in `app.c`.

Sample commands would look like the following:

```
% source /sge/default/common/settings.csh
% cc -I/sge/include -ldrmaa app.c
```

2.23.1.4 Running Your C Application

To run your compiled DRMAA application, verify the following:

The $SGE_ROOT/lib/$SGE_ARCH directory must be included in the library search path (LD_LIBRARY_PATH on the Solaris Operating Environment and Linux). The $SGE_ROOT/lib/$SGE_ARCH directory is not included automatically when you set your environment using the settings.sh or settings.csh files.

You must be logged into a machine that is a Grid Engine submit host. If the machine is not a Grid Engine submit host, all DRMAA function calls will fail, returning DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE.

2.23.1.5 C Application Examples

The following examples illustrate some application interactions that use the C language bindings. You can find additional examples on the "**How To" section of the Grid Engine Community Site**.

Example - Starting and Stopping a Session

Every call to a DRMAA function returns an error code. If everything goes well, that code is DRMAA_ERRNO_SUCCESS. If an error occurs, an appropriate error code is returned.

Every DRMAA function also takes at least two parameters:

- A string to populate with an error message in case of an error
- An integer representing the maximum length of the error string

On line 8, the example calls drmaa_init(). This function sets up the DRMAA session and must be called before most other DRMAA functions. Some functions, like drmaa_get_contact(), can be called before drmaa_init(), but these functions only provide general information. Any function that performs an action, such as drmaa_run_job() or drmaa_wait() must be called after drmaa_init() returns. If such a function is called before drmaa_init() returns, it will return the error code DRMAA_ERRNO_NO_ACTIVE_SESSION.

The dmraa_init() function creates a session and starts an event client listener thread. The session is used for organizing jobs submitted through DRMAA, and the thread is used to receive updates from the queue master about the state of jobs and the system in general. Once drmaa_init() has been called successfully, the calling application must also call drmaa_exit() before terminating. If an application does not call drmaa_exit() before terminating, the queue master might be left with a dead event client handle, which can decrease queue master performance.

At the end of the program, on line 17, drmaa_exit() cleans up the session and stops the event client listener thread. Most other DRMAA functions must be called before drmaa_exit(). Some functions, like drmaa_get_contact(), can be called after drmaa_exit(), but these functions only provide general information. Any function that performs an action, such as drmaa_run_job() or drmaa_wait() must be called before drmaa_exit() is called. If such a function is called after drmaa_exit() is called, it will return the error code DRMAA_ERRNO_NO_ACTIVE_SESSION.

```
01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:    char error[DRMAA_ERROR_STRING_BUFFER];
06:    int errnum = 0;
07:
08:    errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
09:
10:    if (errnum != DRMAA_ERRNO_SUCCESS) {
```

```
11:       fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
12:       return 1;
13:    }
14:
15:    printf("DRMAA library was started successfully\n");
16:
17:    errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
18:
19:    if (errnum != DRMAA_ERRNO_SUCCESS) {
20:       fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
21:       return 1;
22:    }
23:
24:    return 0;
25: }
```

Example - Running a Job

The following code segment shows how to use the DRMAA C binding to submit a job to Grid Engine. The beginning and end of this program are the same as in the preceding example. The differences are on lines 16 through 59. On line 16, DRMAA allocates a job template. A job template is a structure used to store information about a job to be submitted. The same template can be reused for multiple calls to `drmaa_run_job()` or `drmaa_run_bulk_job()`.

On line 22, the `DRMAA_REMOTE_COMMAND` attribute is set. This attribute tells DRMAA where to find the program to run. Its value is the path to the executable. The path can be relative or absolute. If relative, the path is relative to the `DRMAA_WD` attribute, which defaults to the user's home directory. For this program to work, the script `sleeper.sh` must be in your default path.

On line 32, the `DRMAA_V_ARGV` attribute is set. This attribute tells DRMAA what arguments to pass to the executable.

On line 43 , `drmaa_run_job()` submits the job. DRMAA places the id assigned to the job into the character array that is passed to `drmaa_run_job()`. The job is now running as though submitted by `qsub`. At this point, calling `drmaa_exit()` or terminating the program will have no effect on the job.

To clean things up, the job template is deleted on line 54. This frees the memory DRMAA set aside for the job template, but has no effect on submitted jobs.

Finally, on line 61, `drmaa_exit()` is called. The `drmaa_exit()` call is outside of the **if** structure started on line 18 because when `drmaa_init()` is called, `drmaa_exit()` must be called before terminating, regardless of successive commands.

```
01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:    char error[DRMAA_ERROR_STRING_BUFFER];
06:    int errnum = 0;
07:    drmaa_job_template_t *jt = NULL;
08:
09:    errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
10:
11:    if (errnum != DRMAA_ERRNO_SUCCESS) {
12:       fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
13:       return 1;
14:    }
```

```

   errnum = drmaa_allocate_job_template(&amp;jt, error, DRMAA_ERROR_STRING_BUFFER);

   if (errnum != DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "Could not create job template: %s\n", error);
   }
   else {
      errnum = drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, "sleeper.sh",
                                    error, DRMAA_ERROR_STRING_BUFFER);

      if (errnum != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "Could not set attribute \"%s\": %s\n",
                  DRMAA_REMOTE_COMMAND, error);
      }
      else {
         const char *args[2] = {"5", NULL};

         errnum = drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, args, error,
                                              DRMAA_ERROR_STRING_BUFFER);
      }

      if (errnum != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "Could not set attribute \"%s\": %s\n",
                  DRMAA_REMOTE_COMMAND, error);
      }
      else {
         char jobid[DRMAA_JOBNAME_BUFFER];

         errnum = drmaa_run_job(jobid, DRMAA_JOBNAME_BUFFER, jt, error,
                                 DRMAA_ERROR_STRING_BUFFER);

         if (errnum != DRMAA_ERRNO_SUCCESS) {
            fprintf(stderr, "Could not submit job: %s\n", error);
         }
         else {
            printf("Your job has been submitted with id %s\n", jobid);
         }
      } /* else */

      errnum = drmaa_delete_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);

      if (errnum != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "Could not delete job template: %s\n", error);
      }
   } /* else */

   errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);

   if (errnum != DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
      return 1;
   }

   return 0;
}
```

2.23.2 Developing With the Java Language Binding

2.23.2.1 Important Files for the Java Language Binding

To use the DRMAA Java language binding implementation included with Grid Engine, you need to know where to find the important files. The most important file is the DRMAA JAR file `$SGE_ROOT/lib/drmaa.jar`. To compile your DRMAA application, you must include the DRMAA JAR file in your CLASSPATH. The DRMAA classes are documented in the DRMAA Javadoc, located in the `$SGE_ROOT/doc/javadocs` directory. To access the Javadocs, open the file `$SGE_ROOT/doc/javadocs/index.html` in your browser. When you are ready to run your application, you also need the DRMAA shared library, `$SGE_ROOT/lib/$SGE_ARCH/libdrmaa.so`, which provides the required native routines.

2.23.2.2 Importing the DRMAA Java Classes and Packages

To use the DRMAA classes in your application, your classes should import the DRMAA classes or packages. In most cases, only the classes in the org.ggf.drmaa package will be used. You can import these packages individually or using a wildcard package import. In some rare cases, you might need to reference the Grid Engine DRMAA implementation classes found in the `com.sun.grid.drmaa` package. In those cases, you can import the classes individually or you can import all the classes in a given package. The names of the `com.sun.grid.drmaa` classes do not overlap with the `org.ggf.drmaa` classes, so you can import both packages without creating a namespace collision.

2.23.2.3 Compiling Your Java Application

To compile your DRMAA application, you must include the `$SGE_ROOT/lib/drmaa.jar` file in your CLASSPATH. The `drmaa.jar` file will not be included automatically when you set your environment using the `settings.sh` or `settings.csh` files.

2.23.2.4 How to Use DRMAA With NetBeans 5.*x*

To use the DRMAA classes with your NetBeans 5.0 or 5.5 project, follow these steps:

1. Click mouse button 3 on the project node and select Properties.
2. Determine whether your project generates a build file or uses an existing file.

   If your project uses a generated build file:

   1. Select Libraries in the left column.
   2. Click Add Library.
   3. Click Manage Libraries in the Libraries dialog box.
   4. Click New Library in the Library Management dialog box.
   5. Type DRMAA in the Library Name field in the New Library dialog box.
   6. Click OK to dismiss the New Library dialog box.
   7. Click Add JAR/Folder.
   8. Browse to the `$SGE_ROOT/lib` directory in the file chooser dialog box and select the `drmaa.jar` file.
   9. Click Add JAR/Folder to dismiss the file chooser dialog box.

10. Click OK to dismiss the Library Management dialog box.
11. Select the DRMAA library and click Add Library to dismiss the Libraries dialog box.

If your project uses an existing build file:

1. Select Java Sources Classpath in the left column.
2. Click Add JAR/Folder.
3. Browse to the `$SGE_ROOT/lib` directory in the file chooser dialog box and select the `drmaa.jar` file.
4. Click Choose to dismiss the file chooser dialog box.

3. Click OK to dismiss the properties dialog box.
4. Verify that the DRMAA shared library is in the library search path. To run your application from NetBeans, the DRMAA shared library file `$SGE_ROOT/lib/$SGE_ARCH/libdrmaa.so` must be included in the library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The `$SGE_ROOT/lib/$SGE_ARCH` directory is not included automatically when you set your environment using the `settings.sh` or `settings.csh` files. To set up the path for the shared library, perform one of the following:
   - Set up your environment in the shell before launching NetBeans.
   - Add to the `netbeans-root/etc/netbeans.conf` file to set up the environment, such as:

```
# Setup environment for SGE
. $SGE_ROOT/$SGE_CELL/common/settings.sh
SGE_ARCH=`$SGE_ROOT/util/arch`
LD_LIBRARY_PATH=$SGE_ROOT/lib/$SGE_ARCH; export LD_LIBRARY_PATH
```

2.23.2.5 Running Your Java Application

To run your compiled DRMAA application, verify the following:

- The `$SGE_ROOT/lib/$SGE_ARCH` directory must be included in the library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The `$SGE_ROOT/lib/$SGE_ARCH` directory is not included automatically when you set your environment using the `settings.sh` or `settings.csh` files.
- You must be logged into a machine that is a Grid Engine submit host. If the machine is not a Grid Engine submit host, all DRMAA method calls will fail, throwing a `DrmCommunicationException`.

2.23.2.6 Java Application Examples

The following examples illustrate some application interactions that use the Java language bindings. You can find additional examples on the **"How To" section of the Grid Engine Community Site**.

Example - Starting and Stopping a Session

The following code segment shows the most basic DRMAA Java language binding program.

You must have a Session object to do anything with DRMAA. You get the Session object from a SessionFactory. You get the SessionFactory from the static `SessionFactory.getFactory()` method. The reason for this chain is that the `org.ggf.drmaa.* classes` should be considered an immutable package to be used by every DRMAA Java language binding implementation. Because the package is immutable, to load a specific implementation, the SessionFactory uses a system property to find the implementation's session factory, which it then loads. That session factory is then responsible for creating the session in whatever way it sees fit. It should be noted that even though there is a session factory, only one session may exist at a time.

On line 9, `SessionFactory.getFactory()` gets a session factory instance. On line 10, `SessionFactory.getSession()` gets the session instance. On line 13, `Session.init()` initializes the session. "" is passed in as the contact string to create a new session because no initialization arguments are needed.

`Session.init()` creates a session and starts an event client listener thread. The session is used for organizing jobs submitted through DRMAA, and the thread is used to receive updates from the queue master about the state of jobs and the system in general. Once `Session.init()` has been called successfully, the calling application must also call `Session.exit()` before terminating. If an application does not call `Session.exit()` before terminating, the queue master might be left with a dead event client handle, which can decrease queue master performance. Use the `Runtime.addShutdownHook()` method to make sure `Session.exit()` gets called.

At the end of the program, on line 14, `Session.exit()` cleans up the session and stops the event client listener thread. Most other DRMAA methods must be called before `Session.exit()`. Some functions, like `Session.getContact()`, can be called after `Session.exit()`, but these functions only provide general information. Any function that performs an action, such as `Session.runJob()` or `Session.wait()` must be called before `Session.exit()` is called. If such a function is called after `Session.exit()` is called, it will throw a `NoActiveSessionException`.

```
01: package com.sun.grid.drmaa.howto;
02:
03: import org.ggf.drmaa.DrmaaException;
04: import org.ggf.drmaa.Session;
05: import org.ggf.drmaa.SessionFactory;
06:
07: public class Howto1 {
08:    public static void main(String[] args) {
09:       SessionFactory factory = SessionFactory.getFactory();
10:       Session session = factory.getSession();
11:
12:       try {
13:          session.init("");
14:          session.exit();
15:       } catch (DrmaaException e) {
16:          System.out.println("Error: " + e.getMessage());
17:       }
18:    }
19: }
```

Example - Running a Job

The following code segment shows how to use the DRMAA Java language binding to submit a job to Grid Engine. The beginning and end of this program are the same as in the preceding example. The differences are on lines 16 through 24.

On line 16 , DRMAA allocates a JobTemplate. A JobTemplate is an object that is used to store information about a job to be submitted. The same template can be reused for multiple calls to `Session.runJob()` or `Session.runBulkJobs()`.

On line 17, the `RemoteCommand` attribute is set. This attribute tells DRMAA where to find the program to run. Its value is the path to the executable. The path can be relative or absolute. If relative, the path is relative to the `WorkingDirectory` attribute, which defaults to the user's home directory. For more information on DRMAA attributes, see the DRMAA Javadoc. For this program to work, the script `sleeper.sh` must be in your default path.

On line 18, the `args` attribute is set. This attribute tells DRMAA what arguments to pass to the executable. For more information on DRMAA attributes, see the DRMAA Javadoc.

On line 20, `Session.runJob()` submits the job. This method returns the ID assigned to the job by the queue master. The job is now running as though submitted by qsub. At this point, calling `Session.exit()` or terminating the program will have no effect on the job.

To clean things up, the job template is deleted on line 24. This action frees the memory DRMAA set aside for the job template, but has no effect on submitted jobs.

```
01: package com.sun.grid.drmaa.howto;
02:
03: import java.util.Collections;
04: import org.ggf.drmaa.DrmaaException;
05: import org.ggf.drmaa.JobTemplate;
06: import org.ggf.drmaa.Session;
07: import org.ggf.drmaa.SessionFactory;
08:
09: public class Howto2 {
10:    public static void main(String[] args) {
11:       SessionFactory factory = SessionFactory.getFactory();
12:       Session session = factory.getSession();
13:
14:       try {
15:          session.init("");
16:          JobTemplate jt = session.createJobTemplate();
17:          jt.setRemoteCommand("sleeper.sh");
18:          jt.setArgs(Collections.singletonList("5"));
19:
20:          String id = session.runJob(jt);
21:
22:          System.out.println("Your job has been submitted with id " + id);
23:
24:          session.deleteJobTemplate(jt);
25:          session.exit();
26:       } catch (DrmaaException e) {
27:          System.out.println("Error: " + e.getMessage());
28:       }
29:    }
30: }
```

2.24 Using the Accounting and Reporting Console

The optional Accounting and Reporting Console (ARCo) enables you to gather live reporting data from the Grid Engine system and to store the data for historical analysis in the reporting database, which is a standard SQL database.

Raw reporting data is generated by `sge_qmaster`. This raw data is stored in the `$SGE_ROOT/$SGE_CELL/common/reporting` file. The `dbwriter` program reads the raw data in the reporting file and writes it to the SQL reporting database, where it can be accessed by ARCo.

ARCo supports the following SQL database systems:

- PostgreSQL
- Oracle
- MySQL

The dbwriter provides functionality that helps you to manage your database size, by specifying Derived Values and Deletion Rules.

ARCo also provides a web-based tool that contains a set of predefined SQL queries. The predefined queries supplement the most frequent statistical inquiries. You can modify these queries or create your own. To create your queries, you can use either the Simple Query builder (suitable for SQL novices) or the Advanced Query generator. You can display the data in a tabular, graphical, or pivotal form. You can export the data in CVS or PDF form, or store the result for later viewing. You can also use the arcorun utility to run ARCo queries in a batch mode. For information about arcorun, see ARCo Configuration Files and Scripts for `arcorun`. For more information about how to use ARCo, see How to Start ARCo. For information about how to install ARCo, see Installing the Accounting and Reporting Console (ARCo).

If you have multiple clusters, one `dbwriter` installation per cluster is needed, but only one Reporting installation is needed for all clusters.

2.25 Installing the Accounting and Reporting Console (ARCo)

To effectively install the Accounting and Reporting Console, perform the following tasks in the order that they are listed:

2.25.1 Configuring the Database Server

You must properly install and configure the database server before you can install and use ARCo. Specific database installation instructions and configuration settings differ by database vendor.

2.25.2 How to Configure the ARCo Database on MySQL

1. Log in to the MySQL console as a superuser.

```
# mysql -u root -p<password>
```

2. Create user `arco_write` and grant him privileges.

```
mysql> GRANT ALL on *.* to 'arco_write'@'<database_host>' identified by
'<password>' with GRANT OPTION;
mysql> GRANT ALL on *.* to 'arco_write'@'%' identified by '<password>' with
GRANT OPTION;
```

3. Exit the MySQL console.

```
mysql> \q
```

4. Log in to the MySQL console as `arco_write` user.

```
# mysql -u arco_write -p<password>
```

5. Create the accounting and reporting database.

```
mysql> CREATE DATABASE <db_name>;
```

6. Create user `arco_read` and grant him privileges.

```
mysql> GRANT SELECT,SHOW VIEW on <db_name>.* to 'arco_read'@'<database_host>'
identified by '<password>';
mysql> GRANT SELECT,SHOW VIEW on <db_name>.* to 'arco_read'@'%' identified by
'<password>';
```

Note: The user `'arco_read'@'%'` must be created. If the MySQL database host is the same as the host Sun Java Web Console host where ARCo is running, you also need to create the user `'arco_read'@'database_host'`.

---

7. Multi-cluster configuration. If you are configuring databases for multiple clusters, repeat step 5. through 6., changing the `db_name`, or grant the user privileges on all databases.

   ```
   mysql> GRANT SELECT,SHOW VIEW on *.* to 'arco_read'@'<database_host>'
   identified by '<password>';
   mysql> GRANT SELECT,SHOW VIEW on *.* to 'arco_read'@'%' identified by
   '<password>';
   ```

   Since the same set of read and write users is used for all databases, no additional steps are required to perform cross-cluster queries. See the example of a cross-cluster query.

8. Install the dbwriter and reporting software. See How to Install dbwriter and How to Install Reporting.

2.25.3 How to Configure the ARCo Database on PostgresSQL

1. Configure the PostgresSQL database server, as described in How to Configure the PostgresSQL Server.

2. Log in as the database superuser, for example, `postgres`.

   ```
   # su - postgres
   ```

3. Create the database owner name and password. You will need this information when you install the `dbwriter` and ARCo console as described in How to Install dbwriter and How to Install Reporting.

   ```
   > createuser -P arco_write
   Enter password for new user:
   Enter it again:
   Shall the new role be a superuser? (y/n) n
   Shall the new user be allowed to create databases? (y/n) y
   Shall the new user be allowed to create more new users? (y/n) n
   CREATE USER
   ```

4. Create the accounting and reporting database.

   ```
   > createdb -O arco_write arco
   CREATE DATABASE
   ```

5. Create a database user for reading the database.

   ```
   > createuser -P arco_read
   Enter password for new user:
   Enter it again:
   Shall the new role be a superuser? (y/n) n
   Shall the new user be allowed to create databases? (y/n) n
   ```

```
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

6. Grant `arco_write` permissions on default tablespace. The `dbdefinition.xml` explicitly specifies tablespace name in table definition. The `arco_write` must have permissions to create objects in the specified tablespace.

```
> psql

postgres=# GRANT CREATE ON TABLESPACE pg_default TO arco_write;
```

> **Note:** By using tablespaces, an administrator can control the disk layout of a database installation and optimize performance. You can find detailed information on the PostgreSQL tablespaces in the Postgres documentation.

7. After you have set up the database, install the accounting and reporting software. See How to Install dbwriter and How to Install Reporting.

2.25.4 How to Configure the ARCo Database with Multiple Schemas on PostgresSQL

1. Configure the PostgresSQL database server, as described in How to Configure the PostgresSQL Server.
2. Log in as the database superuser, for example, `postgres`.

```
# su - postgres
```

3. Create database user for writing.

```
> createuser -P arco_write_london

 Enter password for new user:
 Enter it again:
 Shall the new role be a superuser? (y/n) n
 Shall the new user be allowed to create databases? (y/n) y
 Shall the new user be allowed to create more new users? (y/n) n
 CREATE USER
```

4. Repeat step 2 for each cluster, changing the user name. For example, if you have a second cluster called denver, you might use `arco_write_denver`.

> **Note:** The user, database, schema names are arbitrary. You are free to use your own, these were chosen for the demonstrative purposes.

5. Create a database user for reading.

```
> createuser -P arco_read_london
 Enter password for new user:
 Enter it again:
 Shall the new role be a superuser? (y/n) n
 Shall the new user be allowed to create databases? (y/n) n
 Shall the new user be allowed to create more new users? (y/n) n
 CREATE USER
```

6. Repeat step 4 for each cluster, changing the user name. For example, if you have a second cluster called denver, you might use `arco_write_denver`.
7. Create the accounting and reporting database.

   ```
   > createdb arco

   CREATE DATABASE
   ```

8. Log in to the accounting and reporting database console.

   ```
   > psql arco

    arco=#
   ```

9. Grant `arco_write_london` permissions on default tablespace. The `dbdefinition.xml` explicitly specifies tablespace name in table definition. The `arco_write` must have permissions to create objects in the specified tablespace.

   ```
   arco=# GRANT CREATE ON TABLESPACE pg_default TO arco_write_london;
   ```

   Note: By using tablespaces, an administrator can control the disk layout of a database installation and optimize performance. You can find detailed information on the PostgreSQL tablespaces in the Postgres documentation available at `http://www.postgresql.org/docs/8.3/static/manage-ag-tablespaces.html`

10. Repeat step 8 for each cluster, changing the user name.
11. Create schemas. The schema name should equal the owner name of the schema. The owner of the schema is the `arco_write_cluster` user.

    ```
    arco=# CREATE SCHEMA arco_write_london AUTHORIZATION arco_write_london;
    CREATE SCHEMA
    ```

    Schema `arco_write_london` owned by user `arco_write_london` was created.

12. Repeat step 10 for each cluster, changing the schema name and owner name.
13. Grant appropriate privileges for users to schemas that they do not own. By default, users cannot access any objects in schemas they do not own. To allow other user access to the schema, the user needs to be granted USAGE privilege on that schema. Grant `arco_read_cluster` the USAGE privilege on the `arco_write_cluster` schema.

    ```
    arco=# GRANT USAGE ON SCHEMA arco_write_london TO arco_read_london;
    GRANT
    ```

14. Repeat step 12 for each cluster, changing the schema name and `arco_read_cluster` name. For example, for a Denver cluster the schema name should be `arco_write_denver` and the user should be `arco_read_denver`.
15. Set search path for ARCo users. In the reporting queries, tables are referred to by unqualified names, which consist of just the table name. The system determines which table is meant by following a search path, which is a list of schemas to look in. The first matching table in the search path is taken to be the one wanted. If there is no match in the search path, an error is reported, even if matching names

exist in other schemas in the database. In a default setup the search path is: $user, public command `SHOW search_path;` can be run to show search path for the currently connected user. If the schemas in step 10. were created using the pattern `schema_name = user_name`, then no additional steps are required for `arco_write_cluster` users. The `arco_read_cluster` needs to be altered.

```
arco=# ALTER USER arco_read_london SET search_path=arco_write_london;
ALTER ROLE
```

16. Repeat step 14 for each cluster, changing the `arco_read_cluster` and the `search_path`.
17. Verify that `search_paths` are set correctly.

    ```
    arco=# SELECT * FROM pg_user;
    ```

    Each `arco_read_cluster` user should have search_path in useconfig column set to the appropriate `arco_write_cluster`. Each `arco_write_cluster` user should have the useconfig field empty, signifying the default `search_path`.
18. Create the cross-cluster user

    Note: In order to perform cross-cluster queries, one user has to be granted SELECT privileges an all the objects in all of the schemas and access these objects using the fully-qualified name, for example `<schema_name>.<table_name>`. For clarity, we will create a new user. However, you can choose any of your existing users. You will need to supply information for this user during the installation of the reporting module. Perform steps 13 - 18 of How to Migrate a PostgreSQL Database to a Different Schema. See also Creating Cross-Cluster Queries.

19. After you have set up the database, install the `dbwriter` and reporting software. See How to Install dbwriter and How to Install Reporting.

2.25.5 How to Configure the MySQL Database Server

The Accounting and Reporting Console uses views. As a result, the console supports MySQL database version 5.0.36 and higher. For more information on the MySQL database software, see the MySQL documentation available at `http://dev.mysql.com/doc/index.html`

2.25.5.1 MySQL Installation Tips

- To start the MySQL server at boot time, copy `support-files/mysql.server` to `/etc/init.d` and link it to both `/etc/rc3.d/S99mysql` and `/etc/rc0.d/K01mysql`. If MySQL is not installed in `/usr/local/mysql`, edit the file to change the basedir and datadir variables.
- Add the full pathname of this directory to your PATH environment variable so that your shell finds the MySQL programs properly.

2.25.5.2 Case Sensitivity in MySQL Database

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory. Because of this the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. Therefore, database and table names are case sensitive in most varieties of UNIX, and not case sensitive in Windows.

1. Download the appropriate MySQL software for your system from `http://www.mysql.com`

   The standard installation directory for UNIX systems is `/usr/local/mysql`. If you install the software into a different directory, you have to change the settings for the scripts provided in the package.

   Note: ARCo is a Java web-based application and needs the Java DataBase Connectivity (JDBC) driver for converting JDBC calls into the network protocol used by the MySQL database. You can download the JDBC driver from `http://www.mysql.com/products/connector`

   ---

2. Create a symbolic link from the installation directory to MySQL.

   ```
   # ln -s $installation_directory/mysql-standard-5.0.26-solaris10-i386 mysql
   ```

   The mysql directory contains several files and subdirectories.

3. Add a login user and group for mysqld.

   ```
   # groupadd mysql
   # useradd -g mysql mysql
   ```

4. Create the MySQL grant tables.

   ```
   # scripts/mysql_install_db --user=mysql
   ```

5. Change the ownership of program binaries to root and ownership of the data directory to the user that you use to run mysqld.

   ```
   # chown -R root .
   # chown -R mysql data
   # chgrp -R mysql .
   ```

6. Configure MySQL server to use InnoDB as the default storage engine. MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both, those that handle transaction-safe tables, and those that handle non-transaction-safe tables. ARCo installation requires the use of transaction-safe tables. Edit the `my.cnf` file and set the option

   ```
   default_storage_engine=innodb
   ```

   Configure other innodb properties such as `innodb_data_home_dir`, `innodb_data_file_path`. For details on InnoDB storage configuration, see `http://dev.mysql.com/doc/refman/5.0/en/innodb-configuration.html`

7. Start the MySQL server.

   ```
   # bin/mysqld_safe --user=mysql &amp;
   ```

8. Assign the root password.

   ```
   # ./bin/mysqladmin -u root password 'new-password'
   # ./bin/mysqladmin -u root -h ${hostname} password 'new-password'
   ```

9. Verify installation. Log in to the MySQL console as a superuser.

```
# mysql -u root -p<password>
```

As a superuser perform these commands:

```
mysql> GRANT ALL on *.* to 'test'@'<database_host>' identified by '<password>'
with GRANT OPTION;
mysql> GRANT ALL on *.* to 'test'@'%' identified by '<password>' with GRANT
OPTION;
```

Log out and log in as the user test.

```
mysql> \q
# mysql -u test -p<password>
```

As the user test perform these commands:

```
mysql> CREATE DATABASE test;
mysql> USE test;
mysql> CREATE TABLE sge_test (x integer, y varchar(50));
mysql> SHOW TABLE STATUS FROM test LIKE 'sge_test';
```

Note: The field 'Engine' should have a value InnoDB.

2.25.6 How to Configure the PostgresSQL Server

Before you configure the database server, you must download, compile and install the PostgreSQL database software and create a user account to own the database processes. Usually, this user is postgres. Add the PostgreSQL bin directory and necessary `LD_LIBRARY_PATH` settings to your environment. You can find detailed information on the PostgreSQL database in the Postgres documentation at `http://www.postgresql.org/docs/8.3/static/index.html`.

1. If you are running Solaris, change the shared memory kernel parameter. The default shared memory kernel parameter on Solaris is not enough to run Postgres. According to the Postgres documentation, the kernel tunables on `/etc/system` must be changed to the following:

```
set shmsys:shminfo_shmmax=0x2000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=256
set shmsys:shminfo_shmseg=256
*** semaphores
set semsys:seminfo_semmap=256
set semsys:seminfo_semmni=512
set semsys:seminfo_semmns=512
set semsys:seminfo_semmsl=32
```

2. Create a home directory for the postgres user. In this example, the home directory is `/space/postgres/data`.

```
% mkdir -p /space/postgres/data
% useradd -d /space/postgres  postgres
% chown postgres /space/postgres/data
% su - postgres
```

3. Continue as described in the PostgreSQL documentation to set up a database.

```
> initdb -D /space/postgres/data

creating directory /space/postgres/data... ok
creating directory /space/postgres/data/base... ok
creating directory /space/postgres/data/global... ok
creating directory /space/postgres/data/pg_xlog... ok
creating directory /space/postgres/data/pg_clog... ok
creating template1 database in /space/postgres/data/base/1... ok
creating configuration files... ok
initializing pg_shadow... ok
enabling unlimited row size for system tables... ok
initializing pg_depend... ok
creating system views... ok
loading pg_description... ok
creating conversions... ok
setting privileges on built-in objects... ok
vacuuming database template1... ok
copying template1 to template0... ok

Success. You can now start the database server using:
   postmaster -D /space/postgres/data
   or
   pg_ctl -D /space/postgres/data -l logfile start
```

4. Make the following changes to the `pg_hba.conf` file. This change permits unrestricted and password free access to the database superuser postgres but requires md5 encrypted passwords for all other database users. Replace nnn.nnn.nnn with your subnet address without the trailing 0. You also can add access rules on a per-host basis by adding similar lines with host IP addresses.

```
# TYPE  DATABASE  USER  IP-ADDRESS     IP-MASK         METHOD
local   all       postgres                             trust
local   all       all                                  md5
# IPv4-style local connections:
#host    all       all   nnn.nnn.nnn.0  255.255.255.0  md5
```

5. Make the following changes to the `postgresql.conf` file, to enable TCP/IP access from other hosts.

```
tcpip_socket = true
max_connections = 40   (increase if necessary)
```

Note: Ensure that the value of `shared_buffers` is at least twice the value of `max_connections`. On PostgreSQL > 8.0 also modify the value of `listen_addresses`.

6. Start the database. In this example, `-i` enables TCP/IP communication, while `-S` is for silent mode, `-D` specifies the data directory.

```
> postmaster -S -i -D /space/postgres/data
```

7. Verify the installation. As the postgres user, try the following commands:

```
% su - postgres
> createuser -P test_user
Enter password for new user:
Enter it again:
Shall the new role be a superuser? (y/n) n
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

```
> createdb -O test_user -E UNICODE test
CREATE DATABASE
```

8. Execute commands as the database super user.

```
> psql test
Welcome to psql 8.3, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit
test=# create table test (x int, y text);
CREATE TABLE
test=# insert into test values (1, 'one');
INSERT 16982 1
test=# insert into test values (2, 'two');
INSERT 16983 1
test=# select * from test;
x |  y
---+------
1 | one
2 | two
(2 rows)
test=# \q
 > psql -U test_user test
Password:
Welcome to psql 8.3, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit
test=>
```

9. After you have successfully tested your database software, set up the PostgresSQL database.
   - If you plan to support one grid cluster, see How to Configure the ARCo Database on PostgresSQL.
   - If you plan to support more than one grid cluster, see How to Configure the ARCo Database with Multiple Schemas on PostgresSQL.
10. After you have set up the database, install the accounting and reporting software. See How to Install dbwriter and How to Install Reporting.

2.25.7 Using the Oracle Database

1. Ask your database administrator for an instance of an Oracle database. You need two database users for this instance, `arco_write` and `arco_read`. The `arco_write` user must be able to create or alter tables, views, and indexes. During the installation of `dbwriter`, the `arco_read` user is granted SELECT privileges on the objects owned by the `arco_write` user and SYNONYMS for these objects are created in the schema of `arco_read` user. The SYNONYMS are created by `arco_read` user, so this user needs to have privilege to create synonyms. Here is an example how these users should be created on Oracle:

   Note: The actual TABLESPACE and QUOTA values might differ.

```
CREATE USER "ARCO_WRITE" PROFILE "DEFAULT" IDENTIFIED BY "<password>"
       DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP" QUOTA 100 M ON
"USERS" ACCOUNT UNLOCK;

CREATE USER "ARCO_READ" PROFILE "DEFAULT" IDENTIFIED BY "<password>"
       DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP" QUOTA 100 M ON
"USERS" ACCOUNT UNLOCK;

GRANT CREATE TABLE, CREATE VIEW, CREATE SESSION TO "ARCO_WRITE"
GRANT CREATE SYNONYM, CREATE SESSION TO "ARCO_READ"
```

2. Multi-cluster configuration. If you have multiple Grid Engine clusters, you will need one pair of users (`arco_write_cluster`, `arco_read_cluster`) for each cluster. You will need to install one `dbwriter` module per cluster, providing one pair of users each time, but only one reporting installation for all the clusters is necessary. During the installation of reporting module, you will provide information for all your clusters (database schemas).
3. Cross-cluster queries. If you intend to perform cross-cluster queries, ask your database administrator to create another user (`multi_read`) and grant him SELECT privileges on all the objects from the other database schemas. You will provide the information for this user during the installation of the reporting module, and use it to connect to database when performing cross-cluster queries. See the example of a cross-cluster query.
4. Ask your database administrator for the connection parameters to the database.
5. Install the `dbwriter` and reporting software. See How to Install dbwriter and How to Install Reporting.

2.25.8 How to Add Authorized ARCo Users

During the installation of the ARCo reporting module, you are asked to enter a list of users who should have write permissions to the ARCo system. Only those users are allowed save modifications on ARCo.

1. Add users to the appropriate file. The list of authorized users is stored in `$SGE_ROOT/$SGE_CELL/arco/reporting/config.xml`.
2. After editing this file, restart the Sun Java Web Console:

   ```
   # smcwebserver restart
   ```

2.25.9 How to Install dbwriter

Before You Begin

Prior to installing `dbwriter`, you must install and configure the following on your ARCo system:

- Grid Engine 6.2 software
- Java Runtime Environment (JRE) version 1.5 or higher
- Database software, as described in Configuring the ARCo Database Server

Steps

1. Extract the accounting and reporting software using either the tar method or the `pkgadd` method.

- If you use the tar method, type the following command:

  ```
  # cd $SGE_ROOT
  ```

  Next, type the following command as one string, with a space between the `-dc` and the path to the tar file:

  ```
  # gunzip -dc <path-to-location-of-file>/sge-6_2-arco.tar.gz | tar xvpf -
  ```

- If you use the `pkgadd` method, type the following command and respond to the script questions:

  ```
  # cd <cdrom_mount_point>/Sun_Grid_Engine_6_2/ARCo/Packages
  # pkgadd -d . SUNWsgeea
  ```

2. As the administrative user, set your environment variables.

   - If you are using a Bourne shell or Korn shell, type the following command:

     ```
     $ . $SGE_ROOT/default/common/settings.sh
     ```

   - If you are using a C shell, type the following command:

     ```
     % source $SGE_ROOT/default/common/settings.csh
     ```

3. Change the global configuration to enable reporting. For details on how to enable reporting, see About Reporting.

   ```
   % qconf -mconf
   <......>
   reporting_params    accounting=true \
    reporting=true flush_time=00:00:15 joblog=true \
    sharelog=00:00:00<......>
   ```

   By default, report variables are not activated. You can use the `qconf` command to enable statistics gathering on specific variables, as shown in the following example:

   ```
   % qconf -me global
   hostname            global
   <......>
   report_variables    cpu,np_load_avg,mem_free,virtual_free
   <......>
   ```

4. Install the `dbwriter` software. For a complete installation example, see Example dbwriter Installation.

   Note: You must install the `dbwriter` software as a root user.

   Note: If you are upgrading from a Grid Engine version that was released before version 6.2, you must run the installations script with option `-upd`. This will remove existing RC scripts.

   Note: If you run the installation script with option `-nosmf`, SMF will not be used and you will have the option to create an RC script.

5. Specify the location of your Grid Engine root directory (`$SGE_ROOT`). See lines 011 to 026 of the Example dbwriter Installation.
6. Specify the names of your Grid Engine cells. See lines 027 through 048 of the Example dbwriter Installation. If you are not planning to support multiple grid clusters with this ARCo installation, you can use the default cell.
7. Specify the location of your Java Software Development Kit. See lines 049 through 054 of the Example dbwriter Installation.
   - Java Software Development Kit version 1.5 or higher is required.
   - If your JAVA_HOME environment variable is set, the script will use that as a default value.
8. Specify whether you want to use an existing `dbwriter` configuration file. See lines 055 through 060 of the Example dbwriter Installation.

   Note: This step only appears if an existing `dbwriter.conf` is detected in `$SGE_ROOT/$SGE_CELL/common`.

   - Because the configuration file differs between versions, you will be prompted for any missing reporting database connection parameters.
   - If you choose to use the existing `dbwriter` configuration file, the installation script will skip to step 13.
9. Specify the basic connection parameters for the reporting database. See lines 061 through 070 of the Example dbwriter Installation.
10. Specify the database user name and password (owner of the database objects). See lines 071 through 074 of the Example dbwriter Installation. The database user must have permissions to create objects in the database. The default user is `arco_write`.
11. Specify the tablespace for tables and indexes. See lines 075 through 079 of the Example dbwriter Installation. If you are using PostgreSQL or Oracle, you must always specify the following tablespaces:
    - For PostgreSQL, the default tablespace is `pg_default`.
    - For Oracle, the default is typically USERS.

    Note: The `arco_write` user must be granted the CREATE privilege on this tablespace. If the `arco_write` user does not have sufficient privileges, the following error message appears:

    ```
    SEVERE: SQL error: ERROR: permission denied for tablespace pg_
    default
    ```

    To grant privileges, login as a superuser and issue the following command in the database console:

    ```
    GRANT CREATE ON TABLESPACE pg_default to arco_write;
    ```

12. Specify the name of the database schema. See lines 080 through 081 of the Example dbwriter Installation. If you are using PostgreSQL or Oracle, you must supply the schema name. The following values apply:

- For PostgreSQL, this value is normally public. For more information on schemas, see How to Configure the ARCo Database with Multiple Schemas on PostgresSQL and your database manual.
- For Oracle, this value should be the database object's owner name (`arco_write`).

13. Specify the database user name and password (the ARCo web application user). See lines 082 through 090 of the Example dbwriter Installation. The ARCo web application connects to the database using this user, the default is `arco_read`. The user `arco_read` is granted SELECT privilege on the database tables and views.

    Note: You will only be prompted to enter a password if you are using Oracle. The installation connects to the database as this user to create synonyms and thus the password for this user is also needed.

14. Locate the JDBC driver and test the database connection. See lines 091 through 102 of the Example dbwriter Installation.

    - If the corresponding JDBC driver is not found, the following error message appears:

    ```
    Error: jdbc driver org.postgresql.Driver
           not found in any jar file of directory
           /opt/sge62/dbwriter/lib

    Copy a jdbc driver for your database into
    this directory!
    ```

    A JDBC driver is not provided with the distribution. If you need to install the driver, do the following:

    – Copy the appropriate JDBC driver into the `$SGE_ROOT/dbwriter/lib directory`. Use the following drivers:

Database	Drivers
PostgreSQL	`postgresql-8.3-603.jdbc3.jar`
Oracle	`ojdbc14.jar`
MySQL	`mysql-connector-java-5.0.4-bin.jar`

    – After you copy the JAR file to the correct location, press RETURN, and the search repeats.

    - If the database connection test fails, you can repeat the setup procedure.

15. Specify how often the `dbwriter` program should check the reporting file for new data. See lines 103 and 106 of the Example dbwriter Installation.

16. Specify the spool directory. See line 107 the Example dbwriter Installation. The `dbwriter` log and process id (pid) files are stored in this directory.

17. Specify the location of the file that contains the deletion and derived values rules. See line 108 of the Example dbwriter Installation. By default the dbwriter.xml file that contains the deletion and derived values rules is stored in `$SGE_`

ROOT/dbwriter/database/<database_type>/dbwriter.xml. If you move this file to a different location, specify the path to this location. For more information, see Derived Values and Deletion Rules.

18. Set the logging level for the `dbwriter`. See lines 109 through 111 of the Example dbwriter Installation. The following levels are available:
    - WARNING is the least-detailed logging level.
    - FINEST is the most-detailed logging level.
19. Verify the settings. See lines 112 through 129 of the Example dbwriter Installation. If the settings are not correct and you answer n, you are given the option to repeat the setup. If any configuration changes are necessary, do the following:
    - Stop the `dbwriter` (`$SGE_ROOT/$SGE_CELL/common/sgedbwriter stop`)
    - Edit the `dbwriter.conf` file or repeat the installation script
    - Start the `dbwriter` again (`$SGE_ROOT/$SGE_CELL/common/sgedbwriter start`)
20. Check current database model. See lines 130 through 151 of the Example dbwriter Installation. If a newer version of your current database model is necessary, an upgrade is suggested. During the upgrade, the database objects and constraints (tables, views, indexes, primary and foreign keys) are created or updated. Once any necessary upgrades have been completed, the `dbwriter` installation creates two files:
    - A start script `$SGE_ROOT/$SGE_CELL/common/sgedbwriter`
    - A configuration file `$SGE_ROOT/$SGE_CELL/common/dbwriter.conf`.
21. Specify whether `dbwriter` should start at boot time. See lines 152 through 161 of the Example dbwriter Installation. If you choose not to start `dbwriter` automatically, the SMF will not be used. If you choose not to start `dbwriter` automatically and you ran the installations script with the option `-nosmf`, RC scripts will not be created. To start the `dbwriter` manually, use one of the following commands:

```
# /etc/init.d/sgedbwriter start

# $SGE_ROOT/$SGE_CELL/common/sgedbwriter start
```

2.25.10 Example dbwriter Installation

The following example shows a complete `dbwriter` installation. The steps in this example are referred to from the `dbwriter` installation and configuration description at How to Install dbwriter.

Step 4

```
001  % su
002  password:
003  # cd $SGE_ROOT/dbwriter
004  # ./inst_dbwriter
005
006  Welcome to the Grid Engine ARCo dbwriter module 007  installation
007  ------------------------------------------------------------
008  The installation will take approximately 5 minutes
009
010  Hit <RETURN> to continue >
```

Step 5

```
Checking $SGE_ROOT directory
----------------------------

The Grid Engine root directory is:

$SGE_ROOT = /mydiskhome/myuser/sge62

If this directory is not correct (e.g. it may
contain an automounter
prefix) enter the correct path to this directory
or hit <RETURN>
to use default [/mydiskhome/myuser/sge62] >>

Your $SGE_ROOT directory: /mydiskhome/myuser/sge62

Hit <RETURN> to continue >>
```

Step 6

```
Grid Engine cells
-----------------

Grid Engine supports multiple cells.

If you are not planning to run multiple Grid Engine clusters or if you don't
know yet what is a Grid Engine cell it is safe to keep the default cell name

   default

If you want to install multiple cells you can enter a cell name now.

The environment variable

   $SGE_CELL=<your_cell_name>

will be set for all further Grid Engine commands.

Enter cell name [default] >>

Using cell >default<.
Hit <RETURN> to continue >>
```

Step 7

```
Java setup
----------

ARCo needs at least java 1.5

Enter the path to your java installation [/usr/java] >>
```

Step 8

```
Dbwriter configuration file
---------------------------

/mydiskhome/myuser/sge62/default/common/dbwriter.conf found.

Do you want to use the existing dbwriter configuration file? (y/n) [y] >>
```

Step 9

```
Setup your database connection parameters
-----------------------------------------

Enter your database type ( o = Oracle, p = PostgreSQL, m = MySQL ) [] >> o

Enter the name of your oracle database host [] >> ge4

Enter the port of your oracle database [1521] >>

Enter the name of your oracle database [arco] >> arco
```

Step 10

```
Enter the name of the database user [arco_write] >> arco_write

Enter the password of the database user >>
Retype the password >>
```

Step 11

```
The arco_write must have permissions to create objects in the specified tablespace.

Enter the name of TABLESPACE for tables [USERS] >>

Enter the name of TABLESPACE for indexes [USERS] >>
```

Step 12

```
Enter the name of the database schema [arco_write] >> arco_write

```

Step 13

```
The ARCo web application connects to the database with a user which has restricted
access. The name of this database user is needed to grant him access to the sge tables
and must be different from arco_write.
Enter the name of this database user [arco_read] >> arco_read

This user will also create the synonyms for the ARCo tables and views.

Enter the password of the database user >>
Retype the password >>
```

Step 14

```
Database connection test
------------------------

Searching for the jdbc driver oracle.jdbc.driver.OracleDriver
in directory /mydiskhome/myuser/sge62/dbwriter/lib

OK, jdbc driver found

Should the connection to the database be tested? (y/n) [y] >>


Test database connection to 'jdbc:oracle:thin:@ge4:1521:orcl' ... OK
```

Step 15

```
Generic parameters
------------------

Enter the interval between two dbwriter runs in seconds [60] >>
```

Step 16

```
Enter the path of the dbwriter spool directory
[/mydiskhome/myuser/sge62/default/spool/dbwriter]>>
```

Step 17

```
Enter the file with the derived value rules
[/mydiskhome/myuser/sge62/dbwriter/database/oracle/dbwriter.xml] >>
```

Step 18

```
The dbwriter can run with different debug levels
Possible values: WARNING INFO CONFIG FINE FINER FINEST
Enter the debug level of the dbwriter [INFO] >>
```

Step 19

```
All parameters are now collected
--------------------------------

        SGE_ROOT=/mydiskhome/myuser/sge62
        SGE_CELL=default
       JAVA_HOME=/opt/jdk1.5.0 (1.5.0_13)
          DB_URL=jdbc:oracle:thin:@ge4:1521:orcl
         DB_USER=arco_write
       READ_USER=arco_read
      TABLESPACE=USERS
TABLESPACE_INDEX=USERS
       DB_SCHEMA=arco_write
        INTERVAL=60
       SPOOL_DIR=/mydiskhome/myuser/sge62/default/spool/dbwriter
    DERIVED_
FILE=/mydiskhome/myuser/sge62/dbwriter/database/oracle/dbwriter.xml
     DEBUG_LEVEL=INFO

Are these settings correct? (y/n) [y] >>
```

Step 20

```
Database model installation/upgrade
-----------------------------------
Query database version ... no sge tables found
New version of the database model is needed

Should the database model be upgraded to version 8? (y/n) [y] >>

Upgrade to database model version 8 ... Install version 6.0 (id=0) -------
Create table sge_job
Create index sge_job_idx0
.
.
.
Update version table
committing changes
Version 6.2 (id=8) successfully installed
```

```
OK
Create start script sgedbwriter in /mydiskhome/myuser/sge62/default/common

Create configuration file for dbwriter in
/mydiskhome/myuser/sge62/default/common

Hit <RETURN> to continue >>
```

Step 21

```
dbwriter startup script
-----------------------

Do you want to start dbwriter automatically at machine boot?
NOTE: If you select "n" SMF will be not used at all! (y/n) [y] >> n

Creating dbwriter spool directory
/mydiskhome/myuser/sge62/default/spool/dbwriter
starting dbwriter
dbwriter started (pid=4714)
Installation of dbwriter completed
```

2.25.11 How to Install Reporting

Before You Begin

Before you begin, verify that the Sun Java Web Console is installed as explained in How to Install Sun Java Web Console.

> **Note:** On some Linux platforms, you must set `$JAVA_HOME` to point to Java version 1.5 or higher, prior to installing the reporting module.

Steps

1. Change directory to `$SGE_ROOT/reporting`.

   ```
   # cd $SGE_ROOT/reporting
   ```

2. Use the `inst_reporting` script to install the software. For a complete installation example, see Example Reporting Installation.

   ```
   # ./inst_reporting

   Welcome to the Grid Engine ARCo reporting module installation
   ---------------------------------------------------
   The installation will take approximately 5 minutes

   Hit <RETURN> to continue >>
   ```

3. (Optional) Set the path to the Sun Java Web Console. If the installation script cannot find the Sun Java Web Console commands `smcwebserver`, `wcadmin`, and `smwebapp`, the script asks you to add the appropriate path to your $PATH environment variable. The installation script looks in the following places to find the commands:

   1. The $PATH environment variable
   2. Information contained in the packages for Linux or Solaris

   3. The default path (`/opt/sun/webconsole/bin/` for Linux and `/usr/share/webconsole/bin/` for Solaris)

4. Confirm the location of your Grid Engine root directory (`$SGE_ROOT`). See lines 011 through 025 of the Example Reporting Installation.
5. Specify the names of your Grid Engine cells. See lines 026 through 048 of the Example Reporting Installation. If you are not planning to support multiple grid clusters with this ARCo installation, you can use the default cell.
6. Specify the location of your Java Software Development Kit. See lines 049 through 055 of the Example Reporting Installation. Java Software Development Kit version 1.5 or higher is required. If your JAVA_HOME environment variable is set, the script will use that as a default value.
7. Specify the spool directory where all queries and results will be stored. See lines 056 through 063 of the Example Reporting Installation. If this directory does not exist, it will be created for you.
8. Specify the location of your Grid Engine root directory (`$SGE_ROOT`). See lines 011 through 026 of the Example dbwriter Installation.
9. Specify the parameters for the database connection. See lines 064 through 074 of the Example Reporting Installation. If you plan to support more than one grid cluster with this ARCo instance, the next several steps are repeated for each cluster.
10. Specify an accounting and reporting database user name and password. See lines 075 through 079 of the Example Reporting Installation.

    Note: For security reasons, the database user for accounting and reporting should have only read permission (SELECT) for the database tables. Do not use the same database user that was granted CREATE permission on ARCo database.

11. Identify the database schema for ARCo console. See lines 080 through 081 of the Example Reporting Installation. The name of the database schema depends on your database. For a PostgresSQL database, the database schema name should be public. For Oracle, the name of the database schema should the same as the name of the user account which is used by the `dbwriter` (`arco_write`).
12. Identify the name of your grid cluster. See lines 082 through 084 of the Example Reporting Installation. You should use the same name as `$SGE_CLUSTER_NAME`.
13. Confirm cluster database parameters and specify whether to support additional grid clusters. See lines 097 through 106 of the Example Reporting Installation. If you answer yes, the previous several steps are repeated for each cluster.
14. Enter the login names of users who are allowed to store the queries and results. See lines 107 through 112 of the Example Reporting Installation.

    Note: After installation, you can add or delete authorized users by editing the `config.xml` file. See How to Add Authorized ARCo Users.

15. Verify the information. See lines 113 through 119 of the Example Reporting Installation.

16. If a previous version of ARCo is installed, you will be asked to remove it: See lines 120 through 131 of the Example Reporting Installation.
17. Install pre-defined queries. See lines 132 through 157 of the Example Reporting Installation. If the query directory does not exist, it will be created. The example queries will be installed in the spool directory you have specified. Default `/var/spool/arco/queries`. Existing queries will be replaced if you choose Y.
18. Confirm that reporting module is set up. See lines 158 through 176 of the Example Reporting Installation.
19. Confirm that reporting module is registered to the web console and the console starts. See lines 177 through 198 of the Example Reporting Installation. You should see a series of messages that tell you that the ARCo has installed successfully.
20. Check the log file for error or warning messages.

    ```
    # more /var/log/webconsole/console/console_debug_log
    ```

    The accounting and reporting logs are written to the `/var/log/webconsole/console/console_debug_log` file. The default log level is INFO, but you can modify the log level from the command line:

    ```
    # wcadmin add -p -a reporting arco_logging_level=FINE
    ```

    The new log takes effect the next time the console is started or restarted. The possible log levels are WARNING, INFO, FINE, FINER and FINEST.
21. Connect to the Sun Java Web Console by accessing the following URL in your browser and replace the hostname with the name of your master host.

    ```
    https://hostname:6789
    ```
22. Login with your UNIX account.
23. Select the Grid Engine Accounting and Reporting Console.

2.25.12 Example Reporting Installation

The following example shows a complete ARCo reporting installation. The steps in this example are referred to from the ARCo reporting installation and configuration description at How to Install Reporting.

Step 2

```
001   # cd $SGE_ROOT/reporting
002
003   # ./inst_reporting
004
005   Welcome to the Grid Engine ARCo reporting module installation
006   -------------------------------------------------------------
007   The installation will take approximately 5 minutes
008
009   Hit <RETURN> to continue >>
010
```

Step 3

```
011   Checking $SGE_ROOT directory
012   ----------------------------
013
014   The Grid Engine root directory is:
015
```

```
   $SGE_ROOT = /mydiskhome/myuser/sge62

If this directory is not correct (e.g. it may contain an automounter
prefix) enter the correct path to this directory or hit <RETURN>
to use default [/mydiskhome/myuser/sge62] >>

Your $SGE_ROOT directory: /mydiskhome/myuser/sge62

Hit <RETURN> to continue >>

```

Step 4

```
Grid Engine cells
-----------------

Grid Engine supports multiple cells.

If you are not planning to run multiple Grid Engine clusters or if you don't
know yet what is a Grid Engine cell it is safe to keep the default cell name

   default

If you want to install multiple cells you can enter a cell name now.

 The environment variable

   $SGE_CELL=<your_cell_name>

will be set for all further Grid Engine commands.

Enter cell name [default] >>

Using cell >default<.
Hit <RETURN> to continue >>

```

Step 5

```
Java setup
----------

We need at least java 1.5

Enter the path to your java installation [/myhomedisk/SW/jdk1.5.0/sol-amd64]
>>

```

Step 6

```
Spool directory
---------------

In the spool directory the Grid Engine reporting module will
store all queries and results

Enter the path to the spool directory [/var/spool/arco] >>

```

Step 7

```
Cluster Database Setup
```

```
----------------------

Enter your database type ( o = Oracle, p = PostgreSQL, m = MySQL ) [p] >>

Enter the name of your postgresql database host [] >> ge7

Enter the port of your postgresql database [5432] >>

Enter the name of your postgresql database [arco] >>

```

Step 8

```
Enter the name of the database user [arco_read] >>

Enter the password of the database user >>
Retype the password >>

```

Step 9

```
Enter the name of the database schema [public] >>

```

Step 10

```
Enter the name of your cluster
(it is recommended to use the same name as $SGE_CLUSTER_NAME) [ge7:arco:arco_read] >>

```

Step 11

```
Database connection test
------------------------

Searching for the jdbc driver org.postgresql.Driver
in directory /net/gefs.czech/ws/jo195647/sge62/reporting/WEB-INF/lib

OK, jdbc driver found

Should the connection to the database be tested? (y/n) [y] >>

Test database connection to 'jdbc:postgresql://ge7:5432/arco' ... OK

```

Step 12

```
DB parameters are now collected
-------------------------------
    CLUSTER_NAME=ge7:arco:arco_read
          DB_URL=jdbc:postgresql://ge7:5432/arco
          DB_USER=arco_read

Are these settings correct? (y/n) [y] >>

Do you want to add another cluster? (y/n) [n] >>

```

== If yes is answered, steps starting with Cluster Database Setup are repeated, so info for next cluster can be entered.

Step 13

```
Configure users with write access
---------------------------------

Users: myuser1
Enter a login name of a user (Press enter to finish) >>

```

Step 14

```
All parameters are now collected
--------------------------------
      SPOOL_DIR=/var/spool/arco
      APPL_USERS=jo195647

  Are this settings correct? (y/n) [y] >>

```

Step 15

```
Grid Engine reporting module already registered at Sun Java Web Console
-------------------------------------------------------------------

The Grid Engine reporting modules can only be installed
if no previous version is registered.

Should the Grid Engine reporting module com.sun.grid.arco_6.2-Maintrunk be unregistered? (y/n) [y] >>

The reporting web application has been successfully undeployed.

Hit <RETURN> to continue >>

```

Step 16

```
Install predefined queries
---------------------------

query directory /var/spool/arco/queries already exists
Copy examples queries into /var/spool/arco/queries
Query AR_Attributes.xml already exists. Overwrite? ( y = yes, n = no, Y = yes to all, N = no to all ) [n]  >> Y
Copy query AR_Attributes.xml ... OK
Copy query AR_Log.xml ... OK
Copy query AR_Reserved_Time_Usage.xml ... OK
Copy query AR_by_User.xml ... OK
Copy query Accounting_per_AR.xml ... OK
Copy query Accounting_per_Department.xml ... OK
Copy query Accounting_per_Project.xml ... OK
Copy query Accounting_per_User.xml ... OK
Copy query Average_Job_Turnaround_Time.xml ... OK
Copy query Average_Job_Wait_Time.xml ... OK
Copy query DBWriter_Performance.xml ... OK
Copy query Host_Load.xml ... OK
Copy query Job_Log.xml ... OK
Copy query Number_of_Jobs_Completed_per_AR.xml ... OK
Copy query Number_of_Jobs_completed.xml ... OK
Copy query Queue_Consumables.xml ... OK
Copy query Statistic_History.xml ... OK
Copy query Statistics.xml ... OK
Copy query Wallclock_time.xml ... OK

```

```
== if 'n' or 'N' is selected the queries will not be updated (not recommended)
```

Step 17

```
ARCo reporting module setup
---------------------------

Found a previous installed version of the ARCo reporting
modules at /ws/jo195647/sge62/default/arco

Remove directory /ws/jo195647/sge62/default/arco/reporting? (y/n) [y] >>

directory /ws/jo195647/sge62/default/arco/reporting removed
Copying ARCo reporting file into /ws/jo195647/sge62/default/arco/reporting

Setting up ARCo reporting configuration file. After registration of
the ARCo reporting module at the Sun Java Web Console you can find
this file at

      /ws/jo195647/sge62/default/arco/reporting/config.xml

Hit <RETURN> to continue >>

```

Step 18

```
Importing Sun Java Web Console 3.0 files into the /ws/jo195647/sge62/default/arco/reporting
-------------------------------------------------------------------------
Imported files to /ws/jo195647/sge62/default/arco/reporting
Created product images in /ws/jo195647/sge62/default/arco/reporting/com_sun_web_ui/images

Hit <RETURN> to continue >>

Registering the Grid Engine reporting module in the Sun Java Web Console
------------------------------------------------------------------
The reporting web application has been successfully deployed.
Set 1 properties for the com.sun.grid.arco_6.2-Maintrunk application.
Set 1 properties for the com.sun.grid.arco_6.2-Maintrunk application.
Set 1 properties for the com.sun.grid.arco_6.2-Maintrunk application.
Creating the TOC file ... OK

Hit <RETURN> to continue >>

Restarting Sun Java Web Console
-------------------------------
Restarting Sun Java(TM) Web Console Version 3.0.2 ...
The console is running
Grid Engine  ARCo reporting successfully installed
```

2.25.13 How to Install Sun Java Web Console

Note: Sun Java Web Console 3.0 is installed automatically on Solaris 10 Update 3 or later. To install Sun Java Web Console on an older version of the Solaris operating system, follow these steps.

1. Check whether Sun Java Web Console is already available on your system, as is usually the case for the Solaris 10 software and on newer Solaris 9 versions. As root, you can check using the following command:

```
# smcwebserver -V
Version 3.0.2
```

Note: ARCo for Grid Engine 6.2 software requires Sun Java Web Console 3.0.x.

---

2. If you need to install the console, extract the web console package under a temporary directory.

```
# cd /tmp
# umask 022
# mkdir swc
# cd swc
# tar xvf cdrom_mount_point/N1_Grid_Engine_6_2/SunWebConsole/tar/swc_sparc_
3.0.2.tar
```

3. If you are running SuSE 9.0, create symbolic links for each of the `/etc/rc#.d` directories.

```
# ln -s /etc/rc.d/rc0.d /etc/rc0.d
# ln -s /etc/rc.d/rc1.d /etc/rc1.d
# ln -s /etc/rc.d/rc2.d /etc/rc2.d
```

4. Run the Sun Java Web Console setup script.

```
# ./setup -n
<....>
Installation complete.

Starting Sun(TM) Web Console Version 3.0.2...
See /var/log/webconsole/console/console_debug_log for server logging
information
```

   The web console is installed but not started until after the ARCo console installation. Once the console is installed, you can use the following commands to stop, start, or restart the console at any time:

```
# /usr/sbin/smcwebserver start
# /usr/sbin/smcwebserver stop
# /usr/sbin/smcwebserver restart
```

For more information on the Java Web Console, see the official product documentation.

2.26 Planning the ARCo Installation

Before you install the ARCo software, you must plan how to achieve the results that fit your environment. This section helps you make the decisions that affect the rest of the procedure. Write down your installation plan in a table similar to the following example.

Parameter	Value
sge-root directory	__________________
Database software vendor	__________________
Database user (read access)	__________________
Database user (write access)	__________________
Multi-cluster support?	__________________

2.26.1 Supported Operating Platforms

- Solaris 10, 9, and 8 Operating Systems (SPARC Platform Edition)
- Solaris 10 and 9 Operating Systems (x86 Platform Edition)
- Solaris 10 Operating System (x64 Platform Edition)
- Linux x64, kernel 2.4, 2.6, glibc >= 2.3.2
- Linux x86, kernel 2.4, 2.6, glibc >= 2.3.2

2.26.2 Required Software

For ARCo software to function correctly, you must already have installed the following on your ARCo system:

- Grid Engine 6.2 software
- Java Runtime Environment (JRE) version 1.5
- One of the following database software versions
  - PostgresSQL 8.0 through 8.3
  - MySQL 5.0.36 and higher
  - Oracle 9i or 10g
- Sun Java Web Console version 3.0 and one of the following web browsers:
  - Netscape 6.2 and above
  - Mozilla 1.4 and above
  - Internet Explorer 5.5 and above
  - Firefox 1.0 and above

Note: Sun Java Web Console 3.0 is installed automatically with Solaris 10 Update 3 or later. If you need to install Sun Java Web Console, see How to Install Sun Java Web Console.

2.26.3 Disk Space Recommendations

Table 2–1 Recommended Disk Requirements

Component	Space Needed
ARCo software	100 MB
Sun Java Web Console	-
Database server memory	250 to 750 MB *
Database server disk space	10 GB *

* Your specific database server configuration settings depend on the following:

- Cluster size and number of jobs running on cluster
- Setting of joblog in `reporting_params` (`qconf -mconf`)
- Configured `report_variables` (`qconf -me global`)
- Configuration of `dbwriter` deletion rules (`<sge_root>/dbwriter/database/<database_type>/dbwriter.xml`)

For guidelines about determining specific database needs, see Space Requirements for the ARCo Database on the Open Grid Engine site.

2.26.4 Multi-Cluster Support Overview

If you have multiple Grid Engine clusters, you can log in to one instance of ARCo from which you can run reports on all ARCo instances that use the same database vendor and structure. With the ARCo multi-cluster support, one `dbwriter` instance per `qmaster` is still required, but a single reporting installation is sufficient for all qmasters. During the reporting installation, you can supply separate database parameters, such as database name, database user, database host, database password for each cluster, the only condition being that databases are of the same vendor. Database connections are configured from these parameters, which enables you to run the same queries on separate clusters, while logged in to the single instance of the ARCo reporting.

For the multi-cluster database configuration, you can use any of these database setups:

- A single database with multiple schemas (one per each cluster) on a single DBMS
- Separate databases (one per each cluster) on a single DBMS
- Separate databases on separate DBMS (one per each cluster)

If you are not interested in cross-cluster queries, you can choose any of these setups. However, to run cross-cluster queries, you must configure a single database with multiple schemas (one per each cluster) on a single DBMS.

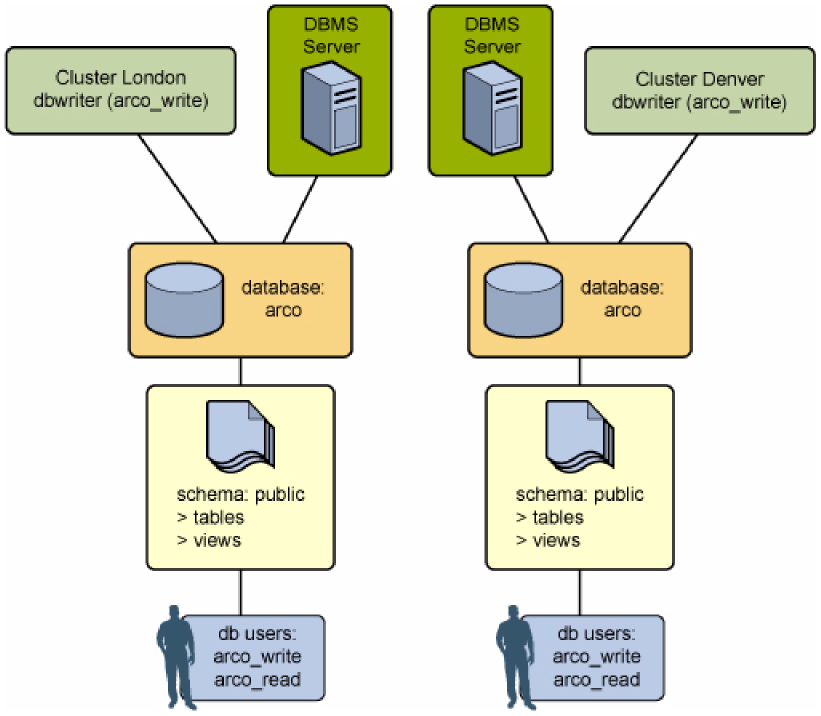
2.26.5 Database Configuration Illustrations

The following diagrams illustrate the supported database configurations. Additional steps, described in, are necessary to configure a PostgresSQL database with separate schemas.

If you want to run cross-cluster queries, use the configuration depicted in Figure 2–14. Otherwise, you can choose either of the other configurations, although Figure 2–13 is slightly preferred.

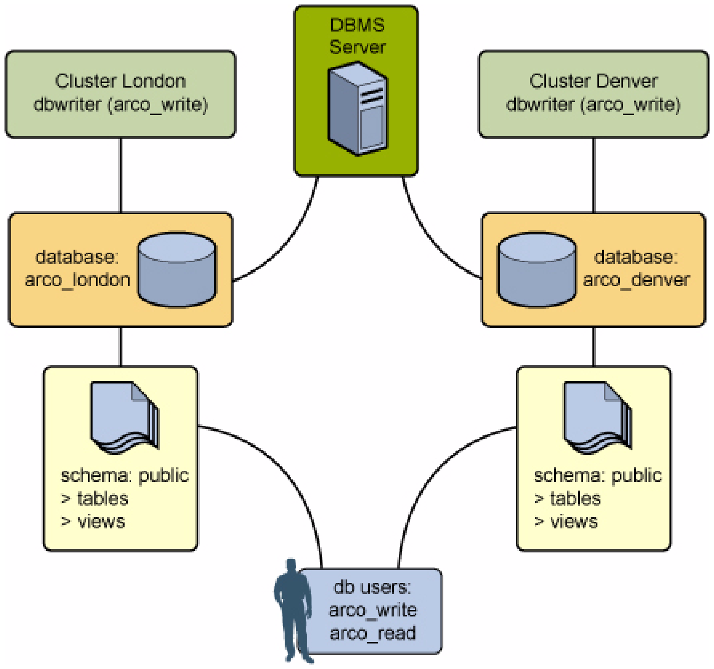
In Figure 2–12, each database is created on a separate Database Management Server (DBMS).

Figure 2–12 Separate Databases on Separate DBMS


DBMS Server
DBMS Server
Cluster London dbwriter (arco_write)
Cluster Denver dbwriter (arco_write)
database: arco
database: arco
schema: public > tables > views
schema: public > tables > views
db users: arco_write arco_read
db users: arco_write arco_read

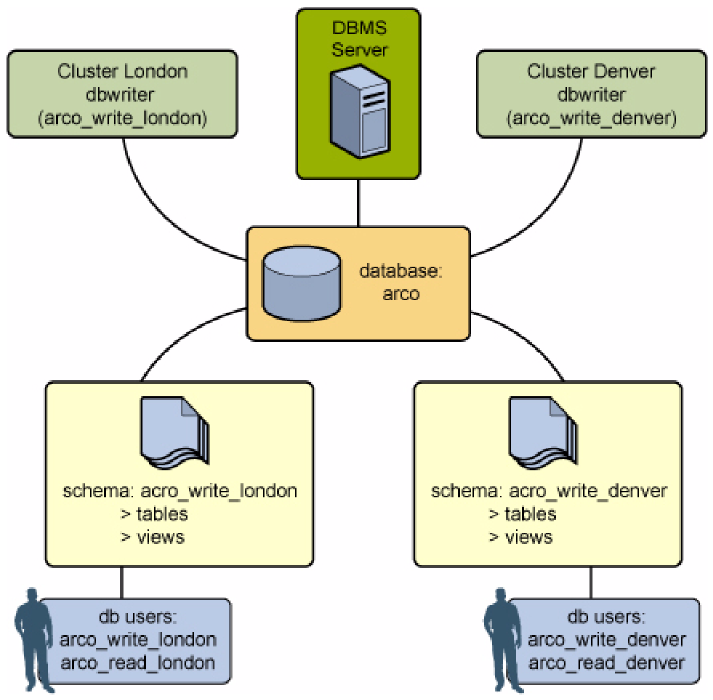
In Figure 2–13, Databases of different names are created on the same DBMS. Only two users are required to access all ARCo databases on the same server, `arco_read` and `arco_write`.

Figure 2–13 Separate Databases on a Single DBMS


DBMS Server
Cluster London dbwriter (arco_write)
Cluster Denver dbwriter (arco_write)
database: arco_london
database: arco_denver
schema: public > tables > views
schema: public > tables > views
db users: arco_write arco_read

In Figure 2–14, only one database is created with multiple schemas (one per each cluster). There are two users for each schema, `arco_write_cluster` and `arco_read_cluster`. The name of the schema should be the same as the name of the owner (`arco_write_cluster`).

Figure 2–14 One Database With Multiple Schemas on a Single DBMS



2.26.6 Schema Overview

A database cluster contains one or more named databases. Any given client connection to the server can access only the data in a single database, the one specified in the connection request. A database can contain one or more named schemas, which in turn contain tables. Schemas also contain other objects, such as views, aliases, indexes and functions. The same object name can be used in different schemas without conflict; for example, both schemas `arco_write_denver` and `arco_write_london` may contain the `sge_job` table.

Unlike databases, schemas are not rigidly separated: a user may access objects in any of the schemas in the database to which the user is connected, if the user has privileges to do so. For user to access objects from a different schema, he needs to be granted SELECT privilege on the objects and access them using the fully-qualified name, for example schema_name.table_name. A user does not need to use the fully-qualified names if accessing objects in its own schema.

Each database handles the schema notion differently.

- Oracle - In Oracle, one schema is created automatically for each user. Because there is a 1-to-1 relationship between a user and a schema, these two terms are often used interchangeably. To perform cross-cluster queries, one designated database user (for example, `multi_read`) needs to be granted SELECT privileges on all the objects (tables, views) from all the other schemas. See Using the Oracle Database.
- PostgreSQL - In PostgreSQL when a table is created without explicitly specifying schema name, it is automatically put into the default schema called 'public'. Each

PostgreSQL database contains such a schema and all users have ALL privilege on that schema. The `dbdefinition.xml` for Postgres allows for explicit specification of schema for table definition. Detailed instructions, described in How to Configure the ARCo Database with Multiple Schemas on PostgresSQL. For more information on schemas, see `http://www.postgresql.org/docs/8.2/static/ddl-schemas.html`

- MySQL - MySQL does not support schemas; the term schema is analogous to database. Command `CREATE SCHEMA` is the same as `CREATE DATABASE`. The implication is that a user can access any object from any database on the same DBMS using a client connection to any single database. If you configure MySQL multiple databases using just one pair of users (`arco_write`, `arco_read`) and grant the privileges as described in How to Configure the ARCo Database on MySQL, you can perform `cross_cluster` queries. You must use the fully-qualified names when accessing objects, for example `database_name.table_name`.

2.27 How to Start ARCo

The accounting and reporting console is installed separately from the Grid Engine software. For details on the installation process, see Installing the Accounting and Reporting Console (ARCo). In addition, you must enable your Grid Engine system to collect reporting information. For details about how to enable the collection of reporting data, see About Reporting.

2.27.1 How to Start the Accounting and Reporting Console

1. From your web browser, type the URL to connect to the Sun Java Web Console.

   In the following example, hostname is the host on which the accounting and reporting software has been installed.

   ```
   https://<hostname>:6789
   ```
2. Log in to your UNIX account.
3. In the Java Web Console main page, select the Accounting and Reporting application.

   Tip: You can also use a link similar to the following example to go directly to the ARCo application from within your web browser: `https://hostname:6789/console/login/Login?redirect_url=%22/reporting/arcomodule/Index%22`.

The Overview page appears. The Query List shows a list of predefined ARCo queries on the selected grid cluster. From the Overview page, you can perform the following tasks:

- To view details about a defined query, click the query Name in the Query List.
- To view the results of any queries that you have run on this cluster, click the Results tab.
- To create a new query, click the New Simple button.
- To create a query by editing the SQL directly, click the New Advanced button.
- To run a defined query, click the circle next to the query name that you want to run, then click the Run button.

- To edit a defined query, click the circle next to the query name that you want to run, then click the Edit button.
- To view information about or run statistics on a different grid cluster, select the cluster from the Cluster menu.

Note: For detailed information about these tasks, see the online help from within the ARCo console.

2.27.2 Creating and Modifying Simple Queries

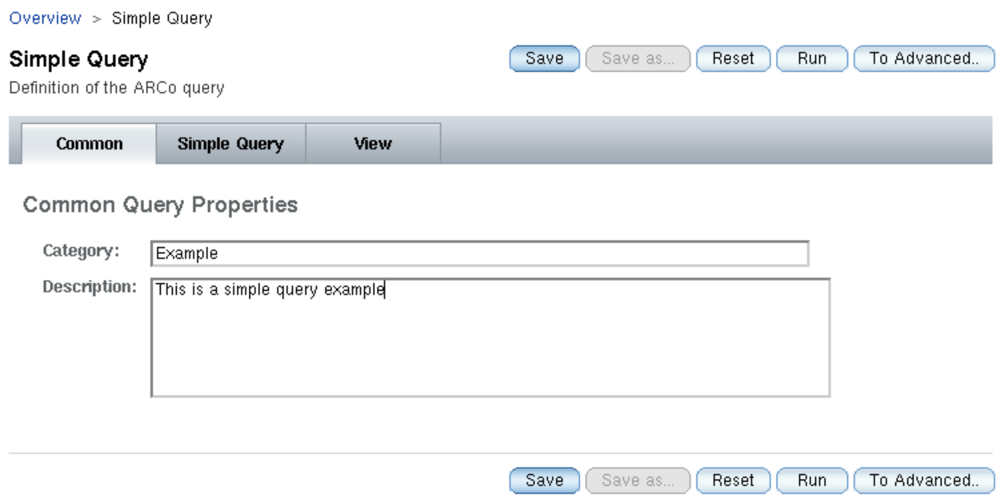
The query defines the data set that you want to retrieve. You can create simple queries for which the system formulates the SQL query string. If you know SQL and you want to write the query yourself, you can create advanced queries as described in Creating and Modifying Advanced Queries.

How to Create a Simple Query

1. In the Query List on the ARCo Overview page, click the New Simple button.

   The Simple Query screen appears showing common information, such as the query category and description. This information is optional.

Figure 2–15 Simple Query



Category: Example

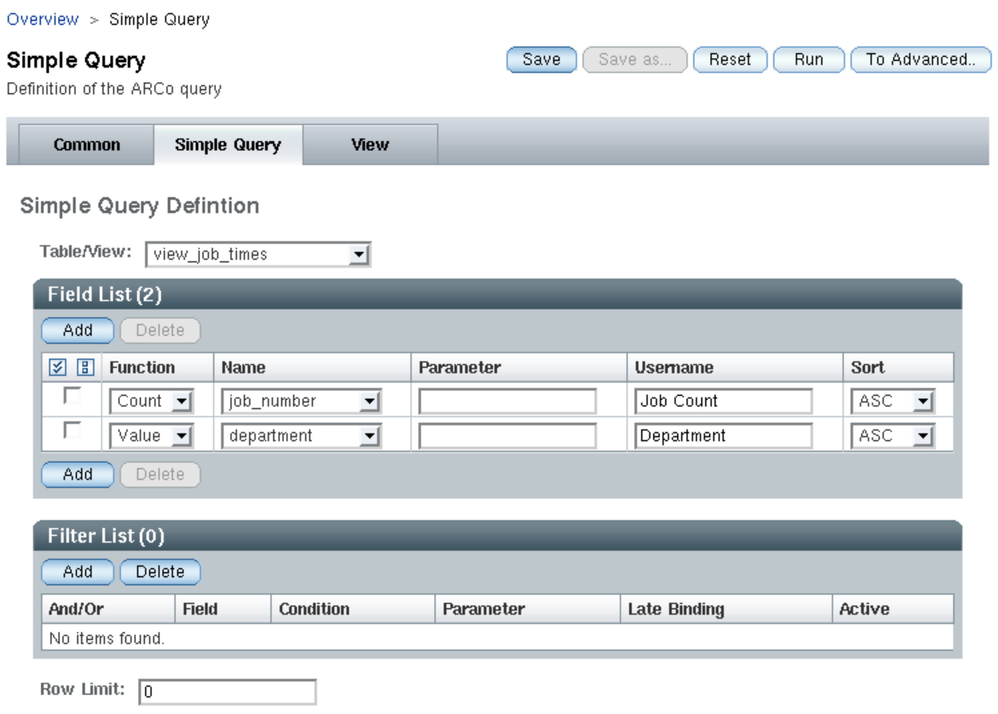
Description: This is a simple query example

2. To define the query, click the Simple Query tab.

   Tip: To define how to display the results of the query, go to the View tab.

3. To choose a database table or view to predefine your query, select from the Table/View list.

Figure 2–16 Simple Query Definition



Simple Query

Definition of the ARCo query

Simple Query Defintion

Table/View: view_job_times

Field List (2)

Function	Name	Parameter	Username	Sort
Count	job_number		Job Count	ASC
Value	department		Department	ASC

Filter List (0)

And/Or	Field	Condition	Parameter	Late Binding	Active
No items found.					

Row Limit: 0

4. To define the fields on which the query is to run, click the Add button in the Fields section.

   - The Function enables you to apply either an aggregate function or a numeric operator to the specified field. Supported values are:

Supported Values	Description
VALUE	Display the current value of the field
SUM	Accumulate the values of the field
COUNT	Count the number of values of the field
MIN	Get the minimum value of the field
MAX	Get the maximum value of the field
AVG	Get the average value of the field

Note: Numeric functions only apply to numeric field values and must be used with a Parameter.

   - The Name is the name of a column in the selected table or view.

- The Parameter is applied when you choose a numeric operator in the Function.
- The Username enables you to provide a more meaningful name to display in the query result.
- Sort enables you to define the sorting order for the field.

5. (Optional) Define Filters.

You must specify at least one field before you can define filters.

- AND/OR is needed for any filter except the first. This setting provides the logical connection to the previous filter condition.
- The Field Name is the name of the field to be filtered. If a field has a user-defined name, that name is shown in the selection list. Otherwise, a generated name is shown.
- The Condition field specifies the operators that are used to filter the values from the database.
- The Parameter field contains a value that is used for filtering the values returned by the query.
- The Parameter field contains a value that is used for filtering the values returned by the query.
- Active enables or disables the filter.

  The following table lists the supported operators.

Condition	Symbol	Description	Number of Parameters	Parameter Usage
Equal	=	Filters the fields that equal the Parameter	1	NA
Not Equal	<>, !=	Filters the fields that do not equal the Parameter	1	NA
Less Than	<	Filters the fields that are less than the Parameter	1	NA
Less Than or Equal	<=	Filters the fields that are less than or equal to the Parameter	1	NA
Greater Than	>	Filters the fields that are greater than the Parameter	1	NA
Greater Than or Equal	>=	Filters the fields that are greater than or equal to the Parameter	1	NA
Null	NA	Filters the fields that are null	0	NA
Not Null	NA	Filters the fields that are not null	0	NA
Between	NA	Filters the fields that are within the specified interval	2	1 AND 100
In	NA	Filters the fields that ar equal to an element of a specified list	1 or more	dep-234, dep-bio, dep-phy

Condition	Symbol	Description	Number of Parameters	Parameter Usage
Like	NA	Filters the fields that match the specified pattern	1	% allows to match any string on any length (including zero length) %bob% will return the only the fields containing the string bob _ allows to match on a single character

6. (Optional) Limit the number of rows to be returned.

   Type the number of rows you want to return in the Row Limit textbox. If the result contains more rows, only the specified number are displayed.

7. Save or run the query.

   To save the query, click Save or Save As.

   To run the query, click Run.

How to Modify a Simple Query

1. Select a query from the list on the Query List screen.
2. Click Edit.

   The selected Simple Query screen displays.

3. Make changes to the Simple Query screen by navigating through the tabs and making your changes as you would when creating a simple query.
4. Save or run the changed query.

   To save the query, click Save or Save As.

   To run the query, click Run.

2.27.3 Creating and Modifying Advanced Queries

Note: You must have previous experience writing SQL queries to use this feature of the accounting and reporting console.

How to Create an Advanced Query

1. In the Query List on the ARCo Overview page, click the New Advanced button.

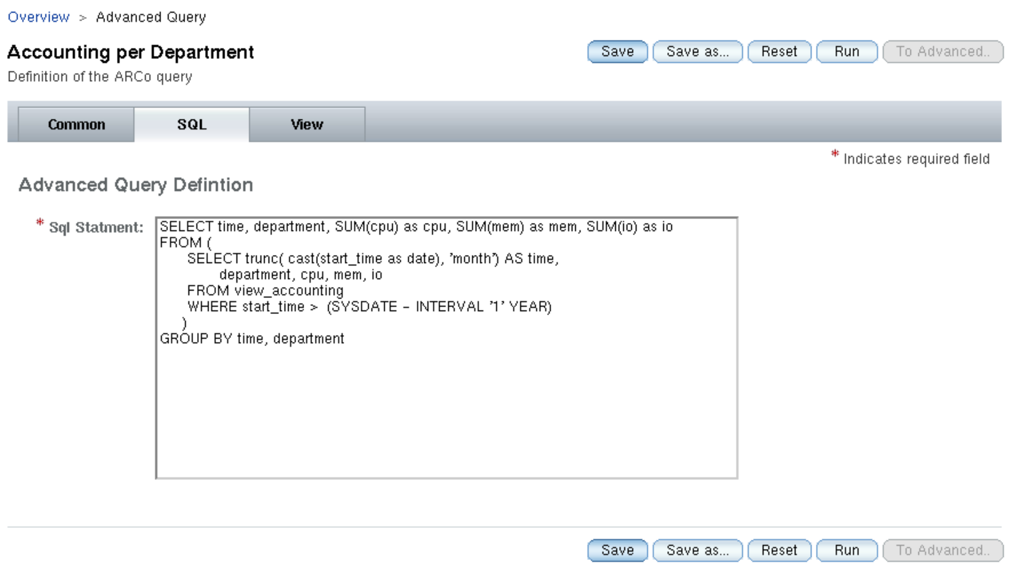
   The Advanced Query screen appears showing common information, such as the query category and description. This information is optional.

2. To define the query, click the Simple Query tab.

Tip: To define how to display the results of the query, go to the View tab.

3. Type your SQL query in the field.

Figure 2–17 Advanced Query Definition


Overview > Advanced Query
Accounting per Department
Definition of the ARCo query
Save
Save as...
Reset
Run
To Advanced..
Common
SQL
View
* Indicates required field
Advanced Query Defintion
* Sql Statment:

```
SELECT time, department, SUM(cpu) as cpu, SUM(mem) as mem, SUM(io) as io
FROM (
     SELECT trunc( cast(start_time as date), 'month') AS time,
           department, cpu, mem, io
     FROM view_accounting
     WHERE start_time > (SYSDATE - INTERVAL '1' YEAR)
    )
GROUP BY time, department
```

4. Save or run the query.

   To save the query, click Save or Save As.

   To run the query, click Run.

How to Edit an Advanced Query

1. Select a query from the list on the Query List screen.
2. Click Edit.

   A completed version of the Advanced Query screen displays.
3. Make changes to the SQL query.
4. Save or run the changed query.

   To save the query, click Save or Save As.

   To run the query, click Run.

Latebindings for Advanced Queries

The syntax for the latebindings in advanced queries is:

```
LATEBINDING { <column>;<operator>;<default value> }

   <column>    name of the latebinding
   <operator>  SQL operator (e.g. = < > in .. )
   <value>     default value (e.g. 'localhost' )
```

Example – Latebindings

```
select * from sge_host where LATEBINDING {h_hostname; like; a%}
select * from sge_host where LATEBINDING {h_hostname; in; ('localhost',
'foo.bar')}
```

2.27.4 Configuring the Query Results View

By default, query results display a database table that contains all the requested information. For Simple and Advanced queries, you can add a pie chart, bar chart, or line diagram to that table. You can also change the view of the database table itself.

How to Configure the Query Results View

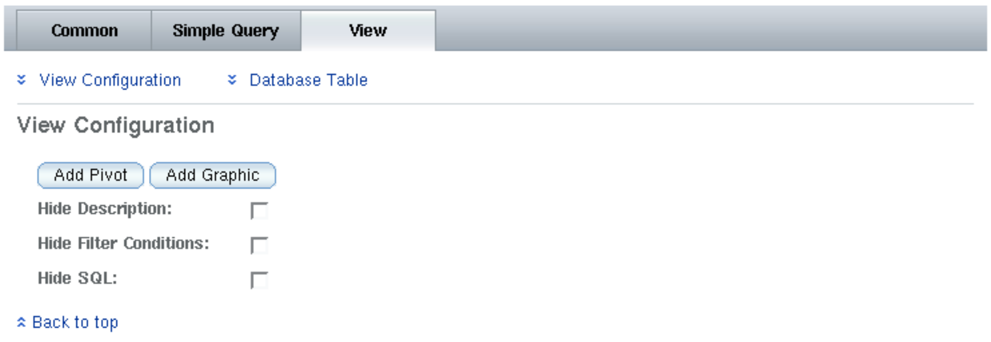
1. To change the view configuration for a query, click the View tab in either the Simple Query or Advanced Query screen.

   To create a view for a saved query:

   - Choose the query from the Query List on the Overview page.
   - Click the Edit button.
   - Click the View tab.

   The current view configuration for the selected query displays.

Figure 2–18 View Configuration



> **Note:** For some queries, only a subset of the possible view selections are meaningful. For example, if you have only two columns to select from, pivot makes no sense.

2. Choose whether to display additional query details.

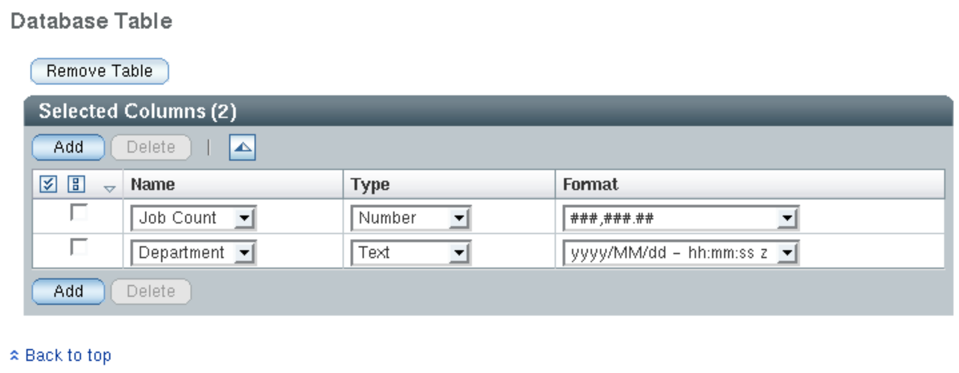
   In the View Configuration section, you can show or hide the following query details:

   - The query description that you entered in the Common tab.
   - The filter conditions or parameters that you defined in the Simple Query.
   - The SQL statement that defines the query, either as assembled by the Simple Query or as you typed it in the SQL tab in the Advanced Query.

3. To configure the table display, click Add Table.

   Choose the columns that you need to display under Name and adjust their Type and Format. The order in which the columns are added will be the order in which the columns are presented. The selections that you make for this report do not affect the filters applied to the data.

Figure 2–19 Database Table


Database Table

Remove Table

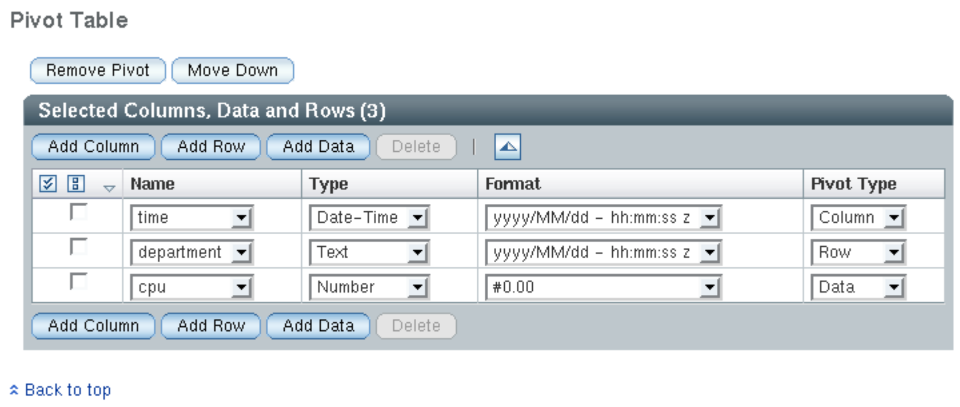
Selected Columns (2)

	Name	Type	Format
☐	Job Count	Number	###,###.##
☐	Department	Text	yyyy/MM/dd - hh:mm:ss z

4. To add a pivot table, click Add Pivot.

Add the pivot column, row, and data entries. Then choose the column Name, Type, and Format. To shift an entry to a different pivot type, select it under Pivot Type.

Figure 2–20 View Pivot Table


Pivot Table

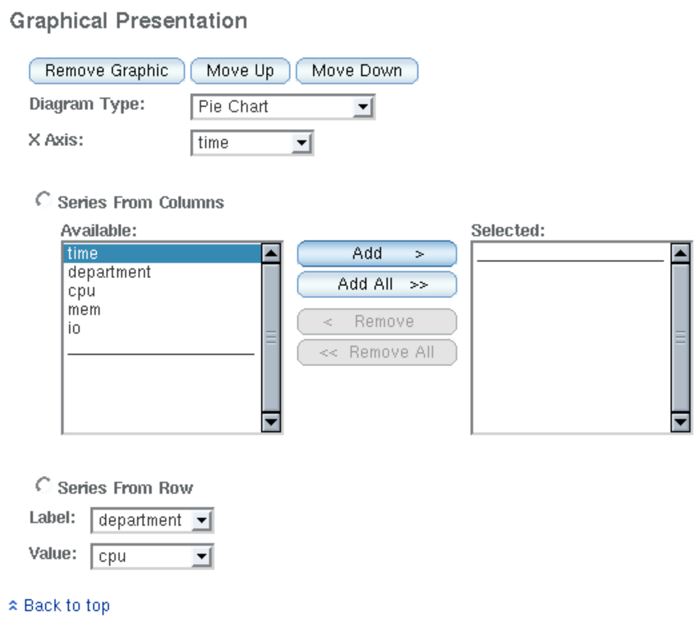
Remove Pivot | Move Down

Selected Columns, Data and Rows (3)

	Name	Type	Format	Pivot Type
☐	time	Date-Time	yyyy/MM/dd - hh:mm:ss z	Column
☐	department	Text	yyyy/MM/dd - hh:mm:ss z	Row
☐	cpu	Number	#0.00	Data

5. To add a graphical view of your data, click Add Graphic.

Figure 2–21 Graphical Presentation of Data



6. Select the diagram type for your graphic.

   You can attach the query data to bar, pie, or line diagram types. The following chart types are available from the Diagram Type menu:

   - Bar Chart
   - Bar Chart (3D
   - Bar Chart Stacked
   - Bar Chart Stacked (3d)
   - Pie Chart, Pie Chart 3D
   - Line Chart
   - Line Chart Stacked Line

     You can choose to display Bar and Pie types with a 3D effect. You can choose to draw stacked Bar and Line diagrams with values on the y-axis summarized.

7. Select the value to display on the X-axis.
8. Decide whether to define the data series based on rows or columns.
   - Series from columns: All column values are added to a series. The name of the series is the column header.
   - Series from rows: All column values define the series. The names of the series is defined by the values of the label column. The values of the series are defined by the value column.
9. Choose specific details as appropriate for your diagram type.

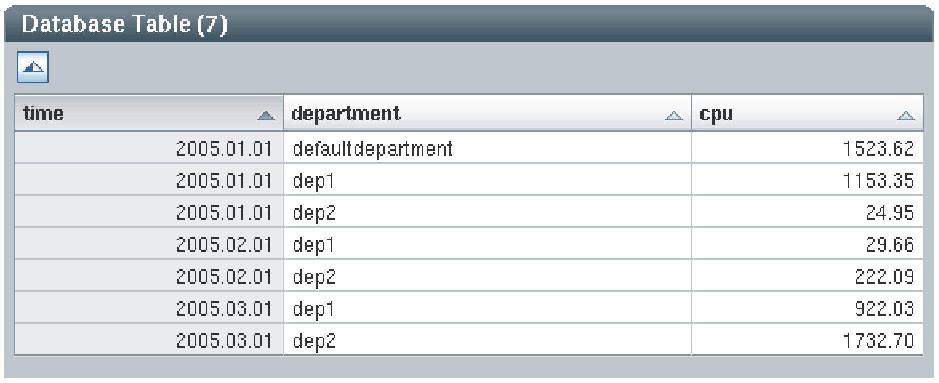
   Because graphic displays are somewhat complex to define, you might find it more useful to look at some examples.
10. Click Save or Save As to save your View configuration to the query.
11. Click Run to run your query.

2.27.5 Examples for Defining Graphical Views

The following two examples show the default view first, followed by the View selections, followed by the graphical result.

Example 1 – Accounting per Department Pie Chart

The query "Accounting per Department" results in a table with the columns: time, department, and cpu.

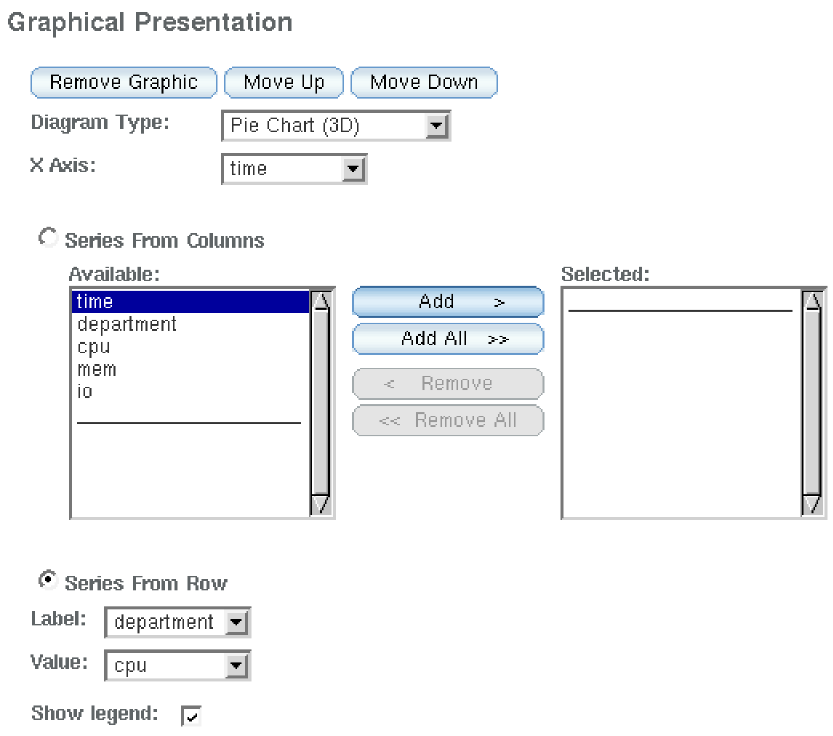


Database Table (7)

time	department	cpu
2005.01.01	defaultdepartment	1523.62
2005.01.01	dep1	1153.35
2005.01.01	dep2	24.95
2005.02.01	dep1	29.66
2005.02.01	dep2	222.09
2005.03.01	dep1	922.03
2005.03.01	dep2	1732.70

Figure 2–22 Accounting per Department Database Table

To display the result in a pie chart, select the following configuration:



Graphical Presentation

Diagram Type: Pie Chart (3D)

X Axis: time

○ Series From Columns

Available: time, department, cpu, mem, io

Selected:

● Series From Row

Label: department

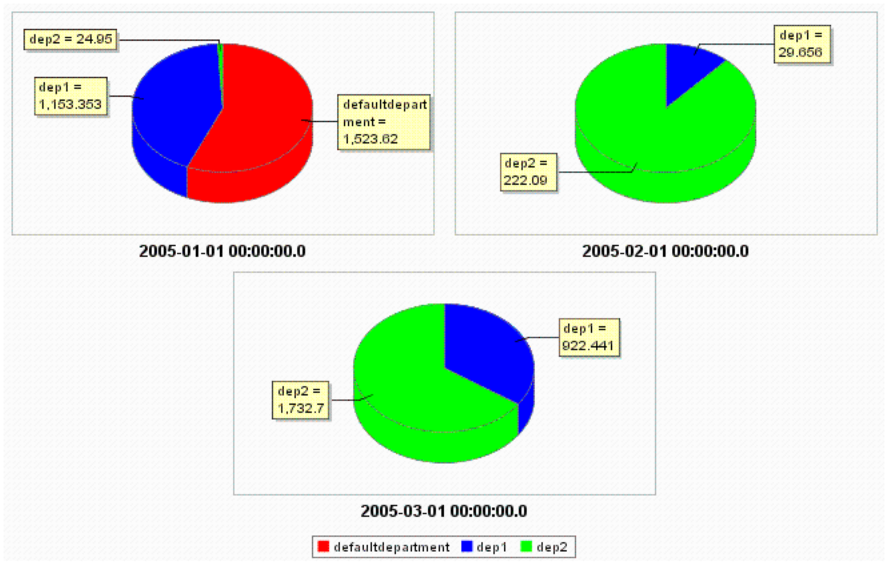
Value: cpu

Show legend: ✓

Figure 2–23 Example of Graphical Presentation

The result will be multiple pie charts, similar to those shown in this figure.

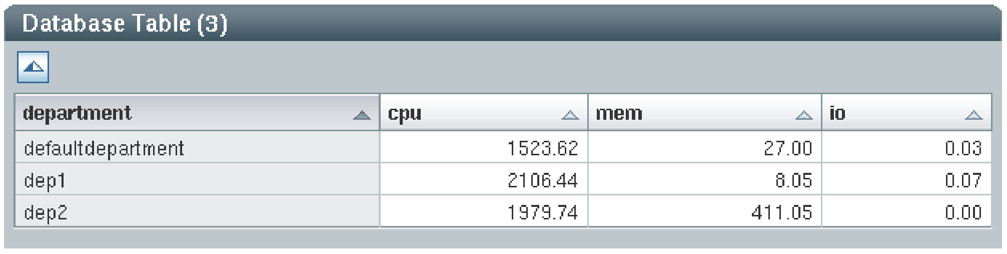
Figure 2–24 Example of Multiple Pie Charts



Example 2 – CPU, Input/Output, and Memory Usage Over All Departments Bar Chart

A query summarizes CPU, IO, and Mem usage over all departments:

Figure 2–25 Example Database Table for Usage



Database Table (3)

department	cpu	mem	io
defaultdepartment	1523.62	27.00	0.03
dep1	2106.44	8.05	0.07
dep2	1979.74	411.05	0.00

To display the results in a bar chart, select the following configuration:

Figure 2–26 Example of Graphical Presentation of Bar Chart

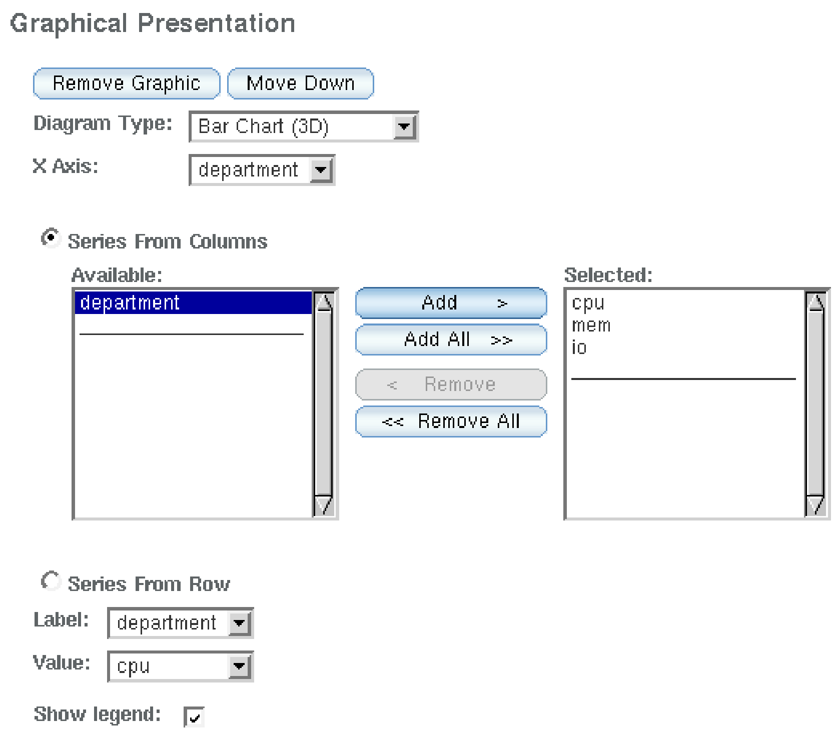


Diagram Type: Bar Chart (3D)
X Axis: department

[✓ Series From Columns]

Available: department

Selected: cpu, mem, io

Series From Row

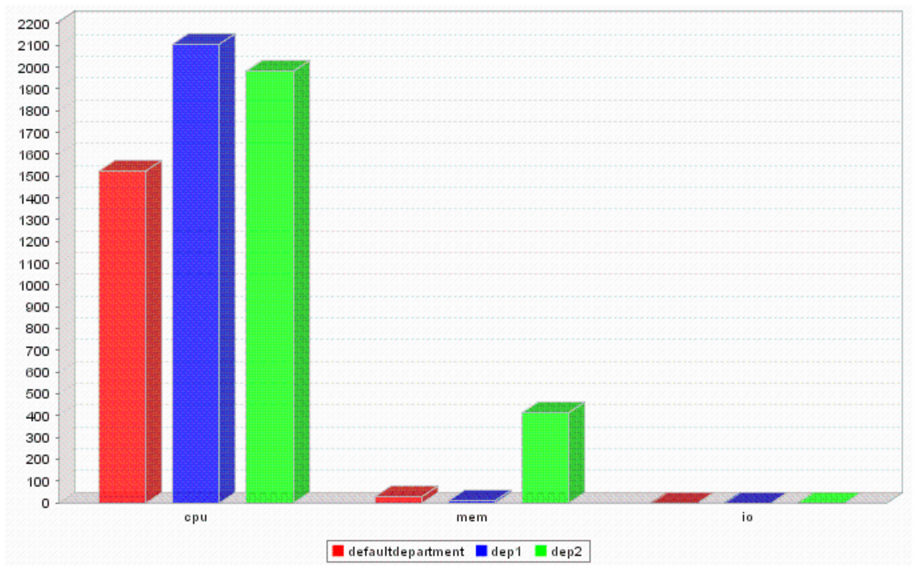
Label: department

Value: cpu

Show legend: ✓

The results will be a bar chart with three bars for each department, similar to the chart shown in this figure.

Figure 2–27 Bar Chart Presentation of Data



2.28 ARCo Configuration Files and Scripts

2.28.1 About dbwriter

The `dbwriter` component writes and deletes the reporting data in the reporting database. It performs the following tasks:

- Reads raw data from the reporting file and writes this raw data to the reporting database.
- Calculates derived values. You can configure which values to calculate, as well as the rules that govern the calculations.
- Deletes outdated data. You can configure how long to keep data.

The `sge_qmaster` component generates the reporting file. You can configure the generation of the reporting file.

When `dbwriter` starts up, it calculates derived values. `dbwriter` also deletes outdated records at startup. If `dbwriter` runs in continuous mode, `dbwriter` continues to calculate derived values and to delete outdated records at hourly intervals, or at whatever interval you specify. See Derived Values and Deletion Rules.

You can specify in a XML file the values that you want to calculate and the records that you want to delete. The path to this file is specified during installation. To change the path to the file, edit the `DBWRITER_CALCULATION_FILE` parameter in the `dbwriter.conf` file.

2.28.1.1 inst_dbwriter Command Options

The `inst_dbwriter` script, used for installing `dbwriter`, is located at `$SGE_ROOT/dbwriter` and supports the following options:

- `-nosmf` - Disables SMF for Solaris 10+ machines. Instead the regular RC scripts are used.
- `-upd` - Removes old RC scripts not containing the `SGE_CLUSTER_NAME` and starts the installation. This option must be used if you are upgrading from version prior to 6.2.
- `-rmrc` - Removes 6.2 RC scripts or SMF service.
- `-h` - Prints usage text to stdout.

If no option is specified, installation is started.

2.28.1.2 dbwriter Configuration Parameters

During `dbwriter` module installation, the following configuration parameters are collected. These parameters are stored in the `$SGE_ROOT/$SGE_CELL/common/dbwriter.conf file`. Changes to the `dbwriter.conf` file require restarting the `dbwriter`.

Table 2–2 dbwriter Configuration Parameters

Parameter	Description	Sample Value
`DBWRITER_USER_PW`	Password of the database user with the write privileges	`password`
`DBWRITER_USER`	Name of the database user with the write privileges; this user will become the owner of the database objects that will be created.	`arco_write`

Table 2–2 (Cont.) dbwriter Configuration Parameters

Parameter	Description	Sample Value
`READ_USER`	Name of the ARCo read user; This user will be granted SELECT privileges on the objects owned by the user specified above, and on Oracle it is also used to create synonyms.	`arco_read`
`DBWRITER_URL`	JDBC URL to database	`jdbc:postgresql://host.domain:5432/arco`
`DB_SCHEMA`	Name of the database schema for the objects	public
`TABLESPACE`	Tablespace used for storing tables	`pg_default`
`TABLESPACE_INDEX`	Tablespace used for storing indexes	`pg_default`
`DBWRITER_CONTINOUS`	Continuous running mode; Default value is true	true
`DBWRITER_INTERVAL`	Interval in s for continuous; Default value is 60 seconds	60
`DBWRITER_DRIVER`	JDBC driver name	`org.postgresql.Driver`
`DBWRITER_REPORTING_FILE`	File name of reporting file	`/myroot/opt/sge62/default/common/reporting`
`DBWRITER_CALCULATION_FILE`	File containing calculation rules	`/myroot/opt/sge62/dbwriter/database/mysql/dbwriter.xml`
`DBWRITER_SQL_THRESHOLD`	The dbwriter writes a warning into the log file if the execution of a single statement takes longer then the `DBWRITER_SQL_THRESHOLD`. The threshold is specified in seconds. If the threshold is 0, no warning will be written.	0
`SPOOL_DIR`	Spool directory of the dbwriter log files and pid file is stored in this directory	`/myroot/opt/sge62/default/spool/dbwriter`
`DBWRITER_DEBUG`	Debug level. Valid values are: WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL	`INFO`

2.28.1.3 sgedbwriter Command Options

The `sgedbwriter` script, used for starting and stopping `dbwriter`, is located at `$SGE_ROOT/$SGE_CELL/common` and supports the following sub-commands:

- `start` - Starts the `dbwriter` as a background process. If no options are supplied, this is the default behavior. You can supply three options with the start command:
  - `-debug` - Start the `dbwriter` in debug mode. This allows you to attach a debugger.
  - `-debug_port port-number` - Specify a port to use for debugging. The default is 8000.
  - `-nosmf` - Force no SMF

- `stop` - Stops the `dbwriter` process.
- `printsetting` - Prints the specified `dbwriter` setting to stdout. The following settings are available:
  - `pid_file` - Prints the default pid file
  - `log_file` - Print the default log file
  - `spool_dir` - Print the default spool directory
- `-h` - Prints usage text to stdout.

The content environment variable JVMARGS is treated as options for the Java virtual machine. If `$JAVA_HOME` is, set the Java virtual machine at `$JAVA_HOME/bin/java` is started.

2.28.2 About Reporting

The reporting file contains the following types of data:

- Host load values and consumable resources
- Queue consumable resources
- Job logging
- Job accounting
- Share-tree usage
- Advance reservation logging
- Advance reservation accounting

Enabling Generation of the Reporting File

When the Grid Engine system is first installed, the reporting file is disabled. To use ARCo, you must enable the reporting file for the cluster. Once enabled, the reporting file will be generated by `sge_qmaster`. By default, the reporting file is located in `$SGE_ROOT/$SGE_CELL/common`. The path to the file is stored in the `DBWRITER_REPORTING_FILE` parameter of the `dbwriter.conf` file.

Once the reporting file is enabled, the `dbwriter` can read raw data from the reporting file and write it to the reporting database.

For complete details about installing and configuring ARCo, see Installing the Accounting and Reporting Console (ARCo).

How to Enable Generation of the Reporting File From the Command Line

To enable reporting from the command line, use the `qconf -mconf` command to set the `reporting_params` attributes, as described in the last step of How to Enable Generation of the Reporting File With QMON.

How to Enable Generation of the Reporting File With QMON

1. To enable reporting with QMON, on the QMON Main Control window click the Cluster Configuration button.
2. On the Cluster Configuration dialog box, select the global host, and click Modify.
3. On the Cluster Settings dialog box, click the Advanced Settings tab.
4. In the Reporting Parameters field, set the following parameters:
   - Set accounting to true. true is the default value.

- Set reporting to true.
- Set `flush_time` to `00:00:15`. `00:00:15` is the default value.
- Set `joblog` to true.
- Set `sharelog` to `00:10:00`. `00:10:00` is the default value.

Reporting Module Configuration Parameters

During reporting module installation, the following configuration parameters are collected. These parameters are stored in the `$SGE_ROOT/$SGE_CELL/arco/reporting/config.xml` file. Changes to the `config.xml` file require restarting the `smcwebserver`.

- The `<database>` element includes several attributes that configure the database connection for the application to use. This element includes two sub-elements and several attributes. Attributes include the following:
  - name
  - host
  - port
  - schema
  - clusterName Sub-elements include the following:
  - `<driver>`
  - `<user>`, which has three attributes:
    - name
    - passwd
    - maxConnections
- The `<appUser>` element identifies each user that is permitted to use the reporting feature. One appUser element is provided for each user that is permitted to use the reporting feature.

  Note: You can edit the `config.xml` file to add additional users. Provide another `appUser` element for each user to add.

- The `<storage>` element defines the storage of ARCo queries and results. This element includes three sub-elements:
  - `<root>` defines the path of the spool directory
  - `<queries>` defines the directory where to store queries
  - `<results>` defines the directory where to store results

Example - Reporting Module Configuration File

The following `config.xml` example illustrates a single cluster configuration. For a multiple cluster configuration, there would be multiple `<database>` tags.

```
<configuration>
 <!--
   Configure the database connection to be used by the application
 -->
 <database name="arco" host="host.domain" port="5432" schema="public"
```

```
clusterName="testsuite">
    <driver type="postgres">
       <javaClass>org.postgresql.Driver</javaClass>
    </driver>
    <user name="arco_read" passwd="ed5sq937d20ecf5c" maxConnections="10"/>
 </database>

 <applUser>
    admin
 </applUser>
 <applUser>
    sgetest1
 </applUser>
 <applUser>
    sgetest2
 </applUser>

 <storage>
   <root>/var/spool/arco</root>
   <queries>queries</queries>
   <results>results</results>
 </storage>
</configuration>
```

2.28.3 Other ARCo Utilities

arcorun

The `arcorun` utility enables you to view and run ARCo queries from the command line. You can view query output in XML (default), CSV, PDF or HTML format. You can also set values for late-binding parameters.

Note: You must run the `arcorun` utility on a host from which the ARCo spooling directory (default: `/var/spool/arco`) is accessible.

Example - Running a Query

A query is run by simply invoking the `arcorun` command with the name of the query as the argument:

```
% $SGE_ROOT/$SGE_CELL/arco/reporting/arcorun Statistics
```

If a query name contains whitespaces you have to put double-quotes around the query name:

```
% $SGE_ROOT/$SGE_CELL/arco/reporting/arcorun "Host Load"
```

updatedb.sh

The `updatedb.sh` utility enables you to preview changes that will be performed on your database. You supply your existing database parameters and choose `y` in the following prompt:

```
Shall we only print all sql statements which will be executed during the upgrade?
(y/n) [y] >>
```

After that, the SQL commands that will be executed during update/upgrade are printed to the `stdout`.

It is not recommended to use this as a substitute for a regular `dbwriter` update/upgrade. If you would choose option `n`, the SQL commands would be executed and only your database definition would be updated, but you would still need to perform regular `dbwriter` re-installation to also update other parts of `dbwriter` that might have changed.

2.29 Creating Cross-Cluster Queries

Note: Prerequisite for performing cross-cluster queries is that you have configured your database with multiple schemas (not necessary for MySQL), and granted one user SELECT privileges an all the objects in all of the schemas. You must use this user when connecting to the database to perform cross-cluster queries.

Although you could JOIN table from one schema with a one from other schema, it might not be useful to do that, as the data comes from separate Grid Engine cluster. However, queries that combine together the results of two or more separate queries could be beneficial.

The syntax is:

```
select_statement1 UNION [ALL] select_statement2
select_statement1 INTERSECT [ALL] select_statement2
select_statement1 EXCEPT [ALL] select_statement2
```

The select_statement is any SELECT statement without an ORDER BY, LIMIT, FOR UPDATE or FOR SHARE clause. These clauses can be appended at the end of all the chained UNION, INTERSECT and EXCEPT queries, which will then be applied to the combined returned result.

UNION effectively appends the result of select_statement1 to the result of `select_statement2` (although there is no guarantee that this is the order in which the rows are actually returned). Furthermore, it eliminates duplicate rows from its result, in the same way as DISTINCT, unless UNION ALL is used.

INTERSECT returns all rows that are both in the result of `select_statement1` and in the result of `select_statement2`. Duplicate rows are eliminated, unless INTERSECT ALL is used

EXCEPT returns all rows that are in the result of `select_statement1` but not in the result of `select_statement2`. Duplicate rows are eliminated, unless EXCEPT ALL is used.

In order to calculate the union, intersection, or difference of two queries, the two queries must be "union compatible", which means that they return the same number of columns and the corresponding columns have compatible data types. The query statements must use the fully qualified object names, that is, `schemaname.tablename`, `schemaname.viewname`, respectively.

Note: In MySQL the syntax would be `database.tablename`, `database.viewname`, respectively.

2.30 Examples

The following two tables are in schema `arco_write_london` and `arco_write_denver`:

2.30.1 Example - arco_write_london.sge_user

```
+------+--------+
| u_id | u_user |
+------+--------+
|    1 | jade   |
|    2 | julie  |
|    3 | john   |
+------+--------+
```

2.30.2 Example - arco_write_denver.sge_user

```
+------+--------+
| u_id | u_user |
+------+--------+
|    1 | john   |
|    2 | david  |
|    3 | rose   |
+------+--------+
```

Connected as the `multi_read` user who has privileges on all the schemas we execute the following queries:

1. SELECT u_user FROM arco_write_london.sge_user UNION SELECT u_user FROM arco_write_denver.sge_user;

   ```
   +--------+
   | u_user |
   +--------+
   | david  |
   | jade   |
   | john   |
   | julie  |
   | rose   |
   +--------+
   ```

2. SELECT u_user from arco_write_london.sge_user UNION ALL SELECT u_user FROM arco_write_denver.sge_user;

   ```
   +--------+
   | u_user |
   +--------+
   | jade   |
   | julie  |
   | john   |
   | john   |
   | david  |
   | rose   |
   +--------+
   ```

3. SELECT u_user FROM arco_write_london.sge_user INTERSECT SELECT u_user FROM arco_write_denver.sge_user;

   ```
   +--------+
   | u_user |
   ```

```
+--------+
| john   |
+--------+
```

4. SELECT u_user FROM arco_write_london.sge_user EXCEPT SELECT u_user FROM arco_write_denver.sge_user;

```
+--------+
| u_user |
+--------+
| jade   |
| julie  |
+--------+
```

2.31 Derived Values and Deletion Rules

2.31.1 Derived Values

At `dbwriter` startup, and in continuous mode once an hour, derived values are calculated. You can configure which values to calculate in an XML file, which is by default in `$SGE_ROOT/dbwriter/database/<database_type>/dbwriter.xml`. `<database_type>` defines the type of database being used; currently, Oracle, PostgreSQL and MySQL are supported. The path to the configuration file is passed to `dbwriter` during installation and is stored in the `dbwriter.conf` file as the value of the parameter `DBWRITER_CALCULATION_FILE`.

The configuration file uses an XML format, and contains rules for both derived values and deleted values (described in the next section).

2.31.1.1 Derived Values Format

The rules for derived values have the following format.

1. The top-level start tag is `<derive>`. The `<derive>` tag has three required attributes:
   - object - Based on this attribute, the derived value is ultimately stored in one of: `sge_host_values`, `sge_queue_values`, `sge_user_values`, `sge_group_values`, `sge_department_values`, `sge_project_values`. The object is one of the following:
     - `host`
     - `queue`
     - `project`
     - `department`
     - `user`
     - `group`
   - interval - The time range specifying how often to calculate the derived values. The time range is one of the following:
     - `day`
     - `hour`
     - `month`

  - year
- variable - This is the name of the variable to hold the calculated data.

2. A second-level start tag describes the way that the value should be derived. This tag must be either `<sql>` or `<auto>`:
   - `<sql>` - This tag contains an SQL statement used for calculating the derived values. The exact syntax of the entries depends upon the type of database being used. The statement must produce the following columns:
     – `time_start` - Together with `time_end`, specifies the time period for the calculated value.
     – `time_end`
     – `value` - The calculated derived value.
   - The SQL statement can contain the following placeholders. `dbwriter` replaces the placeholders for each query, based on a rule:
     – `time_start` - Start time for the query. `dbwriter` searches for the last previously calculated derived value from this rule, and uses this timestamp as the start time for the next query.
     – `time_end` - End time for the query. This timestamp specifies the end of the last passed time range. For example, if the time range is day, and if derived values are calculated at `00:30`, `00:00` is taken as `time_end`.
     – `_key_0,key_1, . . . ,key_n_` - Components of the primary key for the specified object type. For example, the `sge_hosts` table has the primary `h_hostname`. If a rule is processed for the host object type, one query is executed per entry in the `sge_hosts` table, the `_key_0_` placeholder in the SQL statement is replaced by the hostname. The `sge_queue` table has a composed primary key that is made up of `q_qname` and `q_hostname`.
   - `<auto>` - For certain simple derived values, this tag can be used instead of a full SQL query.This tag has two attributes:
     – `function` - which gives the aggregate function to apply to the variable. This can be any function valid for the type of database being used. Some typical functions are AVG, SUM, VALUE, COUNT, MIN or MAX.
     – `variable` - which can be any variable tracked in the following tables: `sge_host_values`, `sge_queue_values`, `sge_user_values`, `sge_group_values`, `sge_department_values`, `sge_project_values` the variable specified must be from the table indicated by the object attribute of the enclosing `<derive>` tag, for example, if the object is host, the variable must be found in `sge_host_values`.
3. Two end tags that match the two start tags.

2.31.1.2 Derived Values Examples

Here is an example of a derivation rule using the `<sql>` tag. The sge_queue table has a composed primary key comprised of `q_qname` and `q_hostname`. For a rule specified for the queue `object_type`, a query will be made for each entry in the `sge_queue` table, the placeholders `_key_0` will be replaced by the queue name and `key_1_` will be replaced by the hostname.

```
<!--average queue utilization per hour-->
 <derive object="queue" interval="hour" variable="h_utilized">
```

```
    <sql>
         SELECT DATE_TRUNC( 'hour', qv_time_start) AS time_start,
                DATE_TRUNC( 'hour', qv_time_start) + INTERVAL '1 hour' AS time_
end,
                AVG(qv_dvalue * 100 / qv_dconfig) AS value
         FROM sge_queue_values
         WHERE qv_variable = 'slots' AND
               qv_parent = (SELECT q_id FROM sge_queue WHERE q_qname = __key_0__
AND q_hostname = __key_1__) AND
               qv_time_start &lt;= '__time_end__' AND
               qv_time_end &gt; '__time_start__'
         GROUP BY time_start
    </sql>
  </derive>
```

Here is an example when the rule above is processed by the `dbwriter`. A query will be made for each entry in the `sge_queue` table, the placeholders `_key_0` will be replaced by the queue name and `key_1_` will be replaced by the hostname. In this example, the results of these queries will be inserted in the `sge_queue_values` table, because object="queue".

```
SELECT DATE_TRUNC('hour', qv_time_start) AS time_start,
       DATE_TRUNC( 'hour', qv_time_start) + INTERVAL '1 hour' AS time_end,
       AVG(qv_dvalue * 100 / qv_dconfig) AS value
FROM sge_queue_values
WHERE qv_variable = 'slots' AND
      qv_parent = (SELECT q_id FROM sge_queue WHERE q_qname = 'all.q' AND q_
hostname = 'my.hostname') AND
      qv_time_start <= '2008-05-21 00:00:00.0' AND
      qv_time_end > '1970-01-01 01:00:00.0'
GROUP BY time_start;
```

Here is an example of a derivation rule using the `<auto>` tag.

```
<derive object="host" interval="day" variable="d_load">
   <auto function="AVG" variable="h_load" />
</derive>
```

2.31.2 Deleting Outdated Records

At `dbwriter` startup, and in continuous mode once an hour, outdated records will be deleted. You can configure how these records are calculated in an XML file, by default in `$SGE_ROOT/dbwriter/database/<database_type>/dbwriter.xml`. `<database_type>` is the type of database being used; currently, Oracle, PostgreSQL and MySQL are supported. The path to the configuration file is passed to dbwriter during installation and is stored in the `dbwriter.conf` file as the value of the parameter `DBWRITER_CALCULATION_FILE`.

2.31.2.1 Deletion Rules Format

The configuration file contains rules for both derived values and deleted values. Deletion rules are of the following format.

- A top-level start tag `<delete>` with three attributes:
  - `scope` - which specifies the type of data to be deleted. Valid entries are:
    * `job`
    * `job_log`
    * `share_log`

    * `host_values`
    * `queue_values`
    * `project_values`
    * `department_values`
    * `user_values`
    * `group_values`
    * `ar`

      Based on this attribute, the values are deleted from the table with the same name with `sge_` prepended.

  - `time_range` - which gives the unit of `time_amount`.
  - `time_amount` - which is the number of units (`time_range`) during which a record is kept.

- An optional second-level start tag `<sub_scope>`, which specifies an additional condition for deletion. A subscope can be configured for all `*_values` scopes and the `share_log` scope.
- One or two end tags matching the two start tags

For certain scopes, a sub-scope can be configured. The sub-scope specifies an additional condition for deletion. A sub-scope can be configured for all `*_values` scopes and for the `share_log` scope. The following rules apply:

- If a sub-scope is configured for a `*_values` rule, it contains a space-separated list of variables to delete.
- If a sub-scope is specified for the `share_log`, it contains a space-separated list of share-tree nodes to delete.
- If sub-scope are used, you should always have a fall-back rule without sub-scope, which will delete all objects that are not explicitly named by the sub-scope.

Here is an example of a delete tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<DbWriterConfig>
  <!-- keep host values for 2 years -->
  <delete scope="host_values" time_range="year" time_amount="2"/>

  <!-- keep queue values one month -->
  <delete scope="queue_values" time_range="month" time_amount="1">
    <sub_scope>slots</sub_scope>
    <sub_scope>state</sub_scope>
  </delete>
</DbWriterConfig>
```

2.31.2.2 Deletion Rules Examples

The following rule indicates that the four variables given in the subscope should be deleted from the table `sge_host_values` after 7 days.

```
<delete scope="host_values" time_range="day" time_amount="7">
        <sub_scope>np_load_avg</sub_scope>
        <sub_scope>cpu<sub_scope>
        <sub_scope>mem_free</sub_scope>
        <sub_scope>virtual_free</sub_scope>
</delete>
```

The following rule says to delete all variables from the table `sge_host_values` after two years:

```
<delete scope="host_values" time_range="year" time_amount="2"/>
```

The following rule says to delete all records for user `fred` after one month:

```
<delete scope="share_log" time_range="month" time_amount="1">
        <sub_scope>fred</sub_scope>
</delete>
```

2.32 ARCo Frequently-Asked Questions

Do I need to re-install database server and create new database every time I update/upgrade ARCo?

No. Generally, you want to keep inserting the data in the same database. You just need to re-install `dbwriter` and Reporting software and during the installation supply your existing database parameters. If a newer version of database model is available, your existing ARCo database model will be updated during the installation of `dbwriter`. See Upgrading ARCO.

Can I restore database backup into a database already containing data?

No. A database backup must only be restored into an empty database. Because ARCo database is a relational database, there are primary key constrains defined on tables. You would run into and SQL error if a primary key (unique identifier), you are trying to restore, already exists in the database.

How do I change the debug level of the dbwriter?

You specify the debug level during the installation of `dbwriter`.

To change the debug level:

1. Stop the `dbwriter`.

   ```
   $SGE_ROOT/$SGE_CELL/common/sgedbwriter stop
   ```

2. Edit the `dbwriter` configuration file (`$SGE_ROOT/$SGE_CELL/common/dbwriter.conf`).

   ```
   #
   # Debug level
   # Valid values: WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL
   #
   DBWRITER_DEBUG=INFO
   ```

3. Start the `dbwriter`.

   ```
   $SGE_ROOT/$SGE_CELL/common/sgedbwriter stop
   ```

In general, you should use the default debug level, which is info. If you use a more verbose debug level, you substantially increase the amount of data output by `dbwriter`.

You can specify the following debug levels:

- `WARNING` - Displays only severe errors and warnings.

- `INFO` - Adds a number of informational messages. INFO is the default debug level.
- `CONFIG` - Gives additional information that is related to `dbwriter` configuration, for example, about the processing of rules.
- `FINE` - Produces more information. If you choose this debug level, all SQL statements run by `dbwriter` are outputted.
- `FINER` - For debugging.
- `FINEST` - For debugging.
- `ALL` - For debugging, displays information for all levels.

How do I verify the version of the installed ARCo database model?

With Grid Engine 6.1 the table `sge_version` was introduced. This table contains the installed versions of the ARCo database model.

Table 2–3 Installed Versions of ARCo Database Model

Column	Type	Description
v_id	integer	version id (with SGE6u1 the version id has been set to 1)
v_version	text	Version text. Contains the Grid Engine version when the database model has been changed to this version
v_time	timestamp	Timestamp of the upgrade to this version

Connect to your database, and as a superuser or the owner of the database objects issue an SQL command:

```
SELECT * FROM sge_version;
```

2.33 ARCo Troubleshooting

PDF Exports in ARCo Require Lots of Memory

Huge reports can result in an OutOfMemoryException when they are exported to PDF.

Workaround - Increase the JVM heap size for the Sun Java Web Console. The following command sets the maximum heap size to 512 MB:

```
# wcadmin add -p -a reporting java.options="... -Xmx512M ..."
```

After you change the heap size, restart the Sun Java Web Console as shown in this command:

```
# smcwebserver restart
```

Problem: Reporting module installation on Red Hat Enterprise Linux

On certain versions of RHEL, while using SJWC 3.0.x or 3.1.x, you might see following error when during the reporting module installation:

```
Registering the SGE reporting module in the Sun Java Web Console
----------------------------------------------------------------
Must have administration privileges to execute this command.
Must have administration privileges to execute this command.
Must have administration privileges to execute this command.
Must have administration privileges to execute this command.
Creating the TOC file ... OK
```

If you run manually the command smcwebserver start, you might see this error:

```
Starting Sun Java(TM) Web Console Version 3.0.2 ...
Exception while starting container instance console: An exception was thrown while
executing
/var/opt/webconsole/domains/console/conf/wcstart nobody
```

These issues are related to the Sun Java Web Console.

Solution: Follow these steps:

Note: The unsuccessful reporting installation has to be run prior to performing these steps.

- Execute these two commands:

  ```
  chmod +x /var/opt/webconsole/domains/console/conf/wcstart
  chmod +x /var/opt/webconsole/domains/console/conf/wcstop
  ```

- Manually edit the `/etc/opt/webconsole/console/service.properties` file and add the following properties (replace the paths with fully qualified names):

  ```
  arco_app_dir=$SGE_ROOT/$SGE_CELL/arco/reporting
  arco_logging_level=INFO
  arco_config_file=$SGE_ROOT/$SGE_CELL/arco/reporting/config.xml
  ```

- Create file reporting.reg in `/etc/opt/webconsole/console/prereg/com.sun.grid.arco_6.2u6` (this is a `regnot` file, which is normally created during deploy).
- Add the following information to the regnot file, which you had created in the previous step (replace paths with fully qualified names):

  ```
  system=false
  debug=0
  context=reporting
  type=webapp
  location=$SGE_ROOT/$SGE_CELL/arco/reporting
  ```

- As a root user, restart the smcwebserver

  ```
  smcwebserver restart
  ```

After this ARCo should function correctly. However, you will still experience the 'Must have administration privileges to execute this command' while executing the wcadmin command.

Problem: No application is registered with this Sun Java(TM) Web Console, or you have no rights to use any applications that are registered.

The above error can happen on some Linux platforms, while using SJWC 3.0.x, if your $JAVA_HOME is not set or is set to a version of Java that is less than 1.5. Another indication of this problem is the absence of the following files: `$SGE_ROOT/$SGE_`

CELL/arco/reporting/WEB-INF/tld and $SGE_ROOT/$SGE_
CELL/arco/reporting/WEB-INF/lib/registrationservlet.jar.

Solution: Follow these steps:

1. Set your $JAVA_HOME variable to at least version 1.5 of the Java software.
2. Reinstall the reporting module.

Problem: SEVERE: SQL error: ERROR: permission denied for tablespace pg_ default

The above SQL error is shown during installation of dbwriter.

Solution: You must always specify the tablespace, unless you are using MySQL. For PostgreSQL, the default tablespace is pg_default. For Oracle, the default is typically USERS. The arco_write user must be granted the CREATE privilege on this tablespace. If the arco_write user does not have the sufficient privileges the above error message appears.

In database console as a superuser issue a command and then repeat the installation:

```
GRANT CREATE ON TABLESPACE pg_default to arco_write;
```

Problem: SEVERE: SQL error: Column 'ju_start_time' in field list is ambiguous

The above error can happen on some minor release versions of MySQL server, namely 5.0.26 or 5.0.27, where MySQL considers, some more complicated queries as syntactically incorrect. Newer versions of MySQL server handle them correctly.

- If you are doing a fresh ARCo installation and not an upgrade, you can safely edit the $SGE_ROOT/dbwriter/database/mysql/dbdefinition.xml file. Remove everything contained between the <version id="6" name="6.1u3"> </version> tags, except the last item. So, the part should look like this:

```
<version id="6" name="6.1u3">
<item>
        <description>Update version table </description>
        <sql>
           INSERT INTO sge_version (v_id, v_version, v_time)
           VALUES(6, '6.1u3', current_timestamp)
        </sql>
     </item>
</version>
```

- If you are upgrading, upgrade your MySQL Server to a higher version, before proceeding with dbwriter installation.

Problem: SEVERE: SQL error: ORA-01031: insufficient privileges

The above error may be caused during the installation of dbwriter, while the synonyms are being created. Because arco_read is the user who uses the synonyms, in ARCo versions > 6.1u4 the synonyms are being created by user arco_read, in the schema of user arco_read. Thus, the user arco_read needs to be granted the privilege to create synonyms. The ARCo users should be granted the following set of privileges:

```
GRANT CREATE TABLE, CREATE VIEW, CREATE SESSION TO "ARCO_WRITE";
GRANT CREATE SYNONYM, CREATE SESSION TO "ARCO_READ";
```

Problem: SEVERE: SQL error: ORA-01749: you may not GRANT/REVOKE privileges to/from yourself

The above SQL error is shown during installation of `dbwriter`.

Solution: During the installation, after the connection test and database version check, you are prompted to enter the name of the user which has a restricted access to the database `arco_read`. The ARCo web application connects to the database using the user `arco_read`, and because this user is not the owner of the database objects it needs to be granted SELECT privilege on those objects. On Oracle synonyms are also created in the schema of the `arco_read` user and thus password for this user is also needed.

If you have entered `arco_write` instead of the `arco_read` user in the prompt below, you would see the errors above. Repeat installation and provide the correct user name.

```
The ARCo web application connects to the database
with a user which has restricted access.
The name of this database user is needed to grant
him access to the sge tables.
This user will create the synonyms for the ARCo
tables and views, so the user's password is needed.

Enter the name of this database user [] >> ARCO_READ

Enter the password of the database user >>
Retype the password >>
```

Note: On PostgreSQL or MySQL, you will not see this error during installation of `dbwriter` but you will not be able to run any queries from the ARCo web application, because the `arco_read` user has not been granted the SELECT privileges on the database objects.

Problem: SEVERE: SQL error: ORA-00955: name is already used by an existing object

The above SQL error is shown during installation of `dbwriter`.

Solution: Same as in the error above.

Problem: The table/view drop-down menu of a simple query definition does not contain any entry, but the tables are defined in the database.

Solution: The problem normally occurs when using Oracle as the database server. During the installation of the reporting module, wrong database schema name has been specified. For Oracle, the database schema name is equal to the name of the database user, which is used by `dbwriter` (the default name is `arco_write`). For Postgres, the database schema is by default public, or if you have configured separate schemas, it is equal to the name of the database user, which is used by `dbwriter`.

Problem: Connection refused.

Solution: The `smcwebserver` might be down. Start or restart the `smcwebserver`.

Problem: The list of queries or the list of results is empty.

Solution: The cause can be any of the following:

- No queries or results are available in the query `/var/spool/arco/queries`, results directory `/var/spool/arco/results` respectively.
- Queries in the XML files are syntactically incorrect. Check the log file `/var/log/webconsole/console/console_debug_log` for error messages from the XML parser.
- User noaccess has no read or write permissions on the query or results directory.

Problem: The list of available database tables is empty.

Solution: The cause can be any of the following:

- The database is down. Start or restart the database.
- No more database connections are available. Increase the number of allowable connections to the database.
- An error exists in the configuration file of the application. Check the configuration for wrong database users, wrong user passwords, or wrong type of database, and then restart the application.

Problem: The list of selectable fields is empty.

Solution: No table is selected. Select a table from the list.

Problem: The list of filters is empty.

Solution: No fields are selected. Define at least one field.

Problem: The sort list is empty.

Solution: No fields are selected. Define at least one field.

Problem: A defined filter is not used.

Solution: The filter may be inactive. Modify the unused filter and make it active.

Problem: The late binding in the advanced query is ignored, but the execution runs into an error.

Solution: The late binding macro has a syntactical error. The syntax for the late binding in advanced query is:

```
LATEBINDING { <column>;<operator>;<default value> }

   <column>    name if the latebinding
   <operator>  a SQL operator (e.g. = < > in .. )
   <value>     default value (e.g. 'localhost' )
```

Example

```
select * from sge_host where LATEBINDING {h_hostname; like; a%}
select * from sge_host where LATEBINDING {h_hostname; in; ('localhost',
'foo.bar')}
```

Problem: The breadcrumb is used to move back, but the login screen is shown.

Solution: The session has timed out. Log in again, or raise the session timeout value for the Sun Java Web Console (SJWC). To increase the session timeout value to 60 minutes, as a superuser, on the host where the (SJWC) is installed, issue this command:

```
# wcadmin add -p -a reporting session.timeout.value=60
```

Problem: The view configuration is defined, but the default configuration is shown.

Solution: The defined view configuration is not set to be visible. Open the view configuration and define the view configuration to be used.

Problem: The view configuration is defined, but the last configuration is shown.

Solution: The defined view configuration is not set to be visible. Open the view configuration and define the view configuration to be used.

Problem: The execution of a query takes a very long time.

Solution: The results coming from the database are very large. Set a limit for the results, or extend the filter conditions.

3 Upgrading ARCO

Note: The ARCo upgrade is in fact re-installation of the `dbwriter` and reporting modules, during which you supply the parameters of your existing database and database users.

During the installation of `dbwriter`, the existing database schema version is checked and updated, if a newer version is available.

During the installation of reporting, the following actions occur:

- Predefined queries in the reporting spool directory (default: `/var/spool/arco`) are overwritten.

  Note: Only the predefined queries will be overwritten, none of your custom queries will be modified.

- The reporting module is unregistered from the Sun Java Web Console and deployed again.

If you specify a different spool directory during re-installation of reporting, you will need to move your custom queries and results from the spool directory of your previous installation to the new directory, so that they appear in the Sun Java Web Console. Before proceeding with the upgrade, read all the steps in:

- How to Upgrade the ARCo Software
- How to Migrate a PostgreSQL Database to a Different Schema

3.1 How to Migrate a PostgreSQL Database to a Different Schema

If you do not plan to perform cross-cluster queries, follow the standard ARCo upgrade procedure. If you have an existing ARCo installation and want to use the multi-cluster features, follow these steps to migrate existing PostgreSQL ARCo databases to the schema configuration.

1. Prepare a database to which to migrate. Follow How to Configure the ARCo Database with Multiple Schemas on PostgresSQL.

   Note: Throughout this section, in the code snippets and accompanying text, the following values need to be replaced by your appropriate names.

- `postgres` - the database superuser
- `arco` - the database you are migrating to and that has schemas configured
- `filename` - path to a file where all output from the database console will be redirected. The postgres user must have write privileges to the file.
- `arco_write_london` - the schema name you are migrating to.
- `arco_read_london` - is the user used by reporting application to access the database; search_path of this user is set to `arco_write_london`.
- `multi_read` - is a a user used to perform cross-cluster queries; it must be able to access all schemas and read all object in the schemas.

2. Restore data from your first database backup file into arco database. See `http://www.postgresql.org/docs/8.1/interactive/backup.html`.

   Note: After restoring a database backup into a new database with schemas, database object are restored into the default public schema. Hence, you need to restore one backup at a time and only after moving objects to a different schema, you can restore the next backup.

3. Change to the database superuser.

```
# su - postgres
```

4. Log in to the arco database.

```
> psql arco
```

5. Change the output display, so that column name headings and row count footer are not shown.

```
arco=# \t
 Showing only tuples.
```

6. Redirect the output of a query to a file. The postgres user must have write privileges to the file.

```
arco=# \o filename
```

7. Execute the following command to generate commands for moving each table to a different schema.

```
arco=# select 'alter table ' || tablename || ' set schema arco_write_london;'
       from pg_tables where schemaname='public';
```

8. Execute the following command to generate commands for moving each view to a different schema.

```
arco=# select 'alter table ' || viewname || ' set schema arco_write_london;'
       from pg_views where schemaname='public';
```

9. Execute the following commands to generate commands for granting `arco_read_london` select privileges on all the database objects in the specified schema.

```
arco=# select 'grant select on ' ||schemaname||'.'||tablename|| ' to arco_read_
london;'
       from pg_tables where schemaname='arco_write_london';
arco=# select 'grant select on ' ||schemaname||'.'||viewname|| ' to arco_read_
london;'
```

```
       from pg_views where schemaname='arco_write_london';
```

10. Reset the `psql` console to the default state.

```
arco=# \o
arco=# \t
```

11. Run all the commands from the created file.

```
arco=# \i <filename>
```

12. Restore the next backup, and follow the steps 3 - 9, changing the schema and user names appropriately.

    Note: Remember to use a different output filename or delete the previous file.

13. Create the `multi_read` user.

```
arco=# CREATE USER multi_read WITH PASSWORD 'your_password';
```

14. Grant `multi_read` usage on all schemas.

```
arco=# GRANT USAGE ON SCHEMA arco_write_london TO multi_read;
```

15. Repeat the previous step for each schema, changing the schema name.
16. Execute steps 5 - 6.

    Note: Remember to use a different output filename or delete the previous file.

17. Execute the following commands to generate commands granting `multi_read` select privilege on all database object in all the schemas.

```
arco=# select 'grant select on ' ||schemaname||'.'||tablename|| ' to multi_
read;'
       from pg_tables where schemaname in ('arco_write_london', 'next_schema_
name', ...);
arco=# select 'grant select on ' ||schemaname||'.'||viewname||' to multi_read;'
       from pg_views where schemaname in ('arco_write_london', 'next_schema_
name', ...);
```

18. Execute steps 10 - 11.
19. Reinstall `dbwriter`.

    Note: If upgrading from version < 6.2, you must run the installations script with option `-upd`. This will remove existing RC scripts. During the installation, point each `dbwriter` to the newly created database with multiple schemas and specify appropriate `arco_write_cluster` user and schema. See How to Install dbwriter.

20. Reinstall reporting.

Note: During the re-installation of the reporting module, enter the required information for all the configured database schemas. See How to Install Reporting.

3.2 How to Upgrade the ARCo Software

1. Ensure that there are no running or pending jobs.
2. Follow information for shutting down the cluster. See Upgrading From a Previous Release of the Grid Engine Software.
3. Ensure that the reporting file has been completely processed by `dbwriter`, so all the job information from the previous Grid Engine installation has been inserted into the database. There should be no reporting or `reporting.processing` file in the `$SGE_ROOT/$SGE_CELL/common directory`.
4. Once the reporting file has been processed, do the following on the `dbwriter` host:
    1. Source the cluster `settings.sh` (or `.csh`) file.
    2. Stop the `dbwriter`

       ```
       > $SGE_ROOT/$SGE_CELL/common/sgedbwriter stop
       ```
5. Finish upgrading Grid Engine. See Upgrading From a Previous Release of the Grid Engine Software.

   Note: After finishing the Grid Engine upgrade, `qmaster` can be started and jobs submitted to minimize the downtime, as long as `dbwriter` is not started.
6. Back up the existing ARCo databases. Refer to your database manuals on how to backup a database.
7. (Optional) If you plan to perform cross-cluster queries and have PostgreSQL, go to How to Migrate a PostgreSQL Database to a Different Schema; otherwise, continue with the next step.

   Note: You might want to migrate your existing PostgreSQL databases under a single one with multiple schemas, even if you do not plan to perform cross-cluster queries, but you simply like to consolidate all under one roof.
8. Reinstall `dbwriter`.

   Note: If upgrading from version < 6.2, you must run the installations script with option `-upd`. This will remove existing RC scripts. See How to Install dbwriter.
9. Reinstall reporting. See How to Install Reporting.

A

Command Line Interface Ancillary Programs

A.1 List of Ancillary Programs

The Grid Engine system provides the following set of ancillary programs:

Table A–1 Ancillary Programs

Program	Description
`qacct`	Extracts arbitrary accounting information from the cluster log file. For more information, see *Oracle Grid Engine Administration Guide* for generating accounting statistics.
`qalter`	Changes the attributes of submitted but pending jobs.
`qconf`	Provides the user interface for cluster configuration and queue configuration. For more information about using QCONF, see *Oracle Grid Engine Administration Guide*.
`qdel`	Enables a user to delete one or more jobs. A manager or operator can delete jobs belonging to any user, while regular users can only delete their own jobs. For more information, see Monitoring and Controlling Jobs.
`qhold`	Holds back submitted jobs from execution.
`qhost`	Displays status information about execution hosts.
`qlogin`	Initiates a login session with automatic selection of a low-loaded, suitable host.
`qmake`	A replacement for the standard UNIX make facility. qmake extends make by its ability to distribute independent make steps across a cluster of suitable machines. For more information, see Parallel Makefile Processing With qmake.
`qmod`	Enables the owner to suspend or enable a queue. All currently active processes that are associated with this queue are also signaled. For more information, see Monitoring and Controlling Queues and Monitoring and Controlling Jobs.
`qmon`	Provides an X Windows Motif command interface and monitoring facility.
`qping`	Checks application status of Grid Engine daemons.
`qquota`	Shows current usage of Grid Engine resource quotas. For more information, see *Oracle Grid Engine Administration Guide* for information about how to monitor resource quota utilization from the command line.

Table A–1 (Cont.) Ancillary Programs

Program	Description
`qrdel`	Deletes Grid Engine advance reservations. For more information about how to configure advance reservations from the command line, see *Oracle Grid Engine Administration Guide*.
`qresub`	Creates new jobs by copying jobs that are running or pending.
`qrls`	Releases jobs from holds that were previously assigned to them, for example, through `qhold`.
`qrsh`	Can be used for various purposes, such as the following: ■ To provide remote execution of interactive applications through the Grid Engine system. `qrsh` is comparable to the standard UNIX facility `rsh`. For more information, see Remote Execution With qrsh. ■ To allow for the submission of batch jobs that, upon execution, support terminal I/O and terminal control. Terminal I/O includes standard output, standard error, and standard input. ■ To provide a submission client that remains active until the batch job finishes. ■ To allow for the Grid Engine software-controlled remote execution of the tasks of parallel jobs.
`qrstat`	Shows the status of Grid Engine advance reservations. For more information about how to configure advance reservations from the command line, see *Oracle Grid Engine Administration Guide*.
`qrsub`	Submits an advance reservation to Grid Engine. For more information about how to configure advance reservations from the command line, see *Oracle Grid Engine Administration Guide*.
`qselect`	Prints a list of queue names corresponding to specified selection criteria. The output of `qselect` is usually sent to other Grid Engine system commands to apply actions on a selected set of queues.
`qsh`	Opens an interactive shell in an `xterm` on a lightly loaded host. Any kind of interactive jobs can be run in this shell. For more information, see How to Submit Interactive Jobs From the Command Line.
`qstat`	Provides a status listing of all jobs and queues associated with the cluster. For more information, see How to Monitor Jobs From the Command Line.
`qsub`	The user interface for submitting batch jobs to the Grid Engine system.
`qtcsh`	A fully compatible replacement for the widely known and used UNIX C shell (`csh`) derivative, `tcsh`. `qtcsh` provides a command shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts through Grid Engine software. For more information see, Transparent Job Distribution With qtcsh.

A.2 User Access to the Ancillary Program

The following table shows the command capabilities that are available to the different user categories:

Table A–2 User Access to Ancillary Programs

Command	Manager	Operator	Owner	User
`qacct`	Full	Full	Own jobs only	Own jobs only
`qalter`	Full	Full	Own jobs only	Own jobs only

Table A–2 (Cont.) User Access to Ancillary Programs

Command	**Manager**	**Operator**	**Owner**	**User**
`qconf`	Full	No system setup modifications	Show only configurations and access permissions	Show only configurations and access permissions
`qdel`	Full	Full	Own jobs only	Own jobs only
`qhold`	Full	Full	Own jobs only	Own jobs only
`qhost`	Full	Full	Full	Full
`qlogin`	Full	Full	Full	Full
`qmod`	Full	Full	Own jobs and owned queues only	Own jobs only
`qmon`	Full	No system setup modifications	No configuration changes	No configuration changes
`qrexec`	Full	Full	Full	Full
`qselect`	Full	Full	Full	Full
`qsh`	Full	Full	Full	Full
`qstat`	Full	Full	Full	Full
`qsub`	Full	Full	Full	Full